UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

UM FRAMEWORK DE PROGRAMAÇÃO BASEADO EM AJAX PARA TRATAMENTO DE SERVIDORES WEB NÃO CONFIÁVEIS

Anderson Lima Muniz Barretto

DISSERTAÇÃO

MESTRADO EM SEGURANÇA INFORMÁTICA

2013

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

UM FRAMEWORK DE PROGRAMAÇÃO BASEADO EM AJAX PARA TRATAMENTO DE SERVIDORES WEB NÃO CONFIÁVEIS

Anderson Lima Muniz Barretto

DISSERTAÇÃO

Projeto orientado pelo Prof. Doutor Alysson Neves Bessani

MESTRADO EM SEGURANÇA INFORMÁTICA

2013

Agradecimentos

Meus primeiros agardecimentos a Deus, que me abençou com a incrível oportunidade da realização deste projeto, tão veemente lutado e desejado por mim. A Ele, que me abençoou também com saúde, sabedoria, paciência e perseverança para que este trabalho chegasse ao fim com êxito, consagro todo o meu louvor.

Aos meus pais, Raymundo e Davina Barretto, que tanto lutaram para me proporcinar educação e saúde de forma a permitir que este momento se realizasse. Por terem acreditado em mim desde o início e me apoiado em todos os momentos da minha vida. Pelo seu amor incondicional e ilimitado, atualmente extensível a minha esposa e filhos.

A minha querida e amada esposa, Monaliza Barretto, pela paciência no entendimento dos meus momentos ausentes para que este trabalho fosse realizado. Pela sua dedicação e abnegação aos seus intereses em prol do meu sucesso. Pelo incomensurável suporte dado a nossa família durante este trabalho.

Aos meus filhos, Anderson Júnior, Gabriel e Andreza Barretto, pelos abraços, sorrisos e ”boa noite, pai”de cada dia, que tanto me renovaram as forças para que finalmente o êxito fosse alcançado neste trabalho.

Ao meu orientador, Profº Alysson Bessani, sem o qual sua oportuna e acertada orientação, este trabalho não seria possível. Obrigado pela sua atenção, paciência e dedicação, proporcionando as correções de rumos necessárias para o sucesso deste trabalho.

Finalmente, os meus agradecimentos ao Exército Brasileiro, que me presenteou com a realização deste projeto, cuja experiência foi intensamente gratificante e proveitosa.

Ao meu querido filho, Gabriel.

Resumo

As aplicações na Web estão cada vez mais sendo utilizadas por organizações e empresas com o objetivo de vender bens e serviços ou prestar informações de interesse público para a sociedade. Nos últimos anos, o surgimento de novas tecnologias para a programação dessas aplicações proporcionou um significativo incremento em suas funcionalidades e possibilidades, permitindo uma interação maior com seus utilizadores bem como uma experiência de uso mais agradável.

Entretanto, a segurança no uso dessas aplicações é baseada na confiança que um utilizador deve ter nos servidores web que hospedam a aplicação, considerando que os mesmos não estão alterando o conteúdo dos dados da aplicação, enviando conteúdo malicioso com o objetivo de instalar malware em seu computador pessoal, roubar-lhe informações pessoais ou quaisquer outras pretensões maliciosas. Adicionalmente, os utilizadores da aplicação web não têm a garantia de que um servidor web não possa utilizar suas credenciais de autenticação para obter e/ou alterar dados armazenados em uma base de dados da qual o mesmo faz uso.

O presente trabalho tem por objetivo desenvolver e avaliar um framework de programação de aplicações web que permita a interação com um conjunto de servidores web não confiáveis garantindo uma interação segura com uma base de dados.

Para tal, o framework foi desenvolvido utilizando a tecnologia AJAX (*Asynchronous JavaScript and XML*), de forma a permitir que o navegador do utilizador possa verificar a integridade e autenticidade dos dados da aplicação web, bem como realizar a mudança de servidor web de forma transparente, tão logo receba conteúdo malicioso de um servidor ou a comunicação com o mesmo ultrapasse um limite de tempo pré-definido. Adicionalmente, uma aplicação web será implementada utilizando tal framework e avaliada quanto a sua fiabilidade e desempenho.

Palavras-chave: Aplicações na Web, AJAX, Segurança Informática, Servidores Web Não Confiáveis, Computação nas Nuvens

Abstract

Web applications are increasingly being used by organizations and companies in order to sell goods and services or provide public information to society. In recent years, the emergence of new technologies for programming web applications resulted in a significant increase in their functionality and possibilities, allowing greater interaction with its users and a more enjoyable user experience.

However, the security of such applications is based on the trust that a user must have in the web servers that host the application, considering that they are not changing the content of the data to be displayed to the users, sending malicious content, in order to install malware on your personal computer, steal personal information or any other malicious intentions. Additionally, users of the web application does not have a guarantee that a web server can not use their authentication credentials to obtain and / or modify data stored in a database from which it makes use.

The purpose of this study is to develop a framework for programming web applications allowing the handling of a set of untrusted web servers ensuring a secure interaction with the application database.

To this end, the framework was developed using AJAX technology (Asynchronous JavaScript and XML), to allow the user's browser verify the authenticity and integrity of web application and switch web server transparently, so soon receive corrupted content from a malicious server or the communication exceeds a timeout. Additionally, a web application was implemented using this framework and evaluated in terms of reliability and performance.

Keywords: Web applications, AJAX, Security, Untrusted Web Servers, Cloud Computer.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

As aplicações web estão atualmente sendo utilizadas para uma grande variedade de atividades: correio eletrônico, redes sociais, comércio eletrônico, prestação de serviços públicos, gestão financeira, dentre outras. Entretanto, a segurança do uso dessas aplicações baseia-se na confiança que os utilizadores têm nos servidores web que hospedam a referida aplicação. Uma vez que existam vulnerabilidades na aplicação ou na sua arquitetura que possam comprometer a integridade e/ou a autenticidade da mesma, ou até mesmo que, inadvertidamente, tais servidores passem a se comportar de maneira maliciosa com o objetivo de roubar informações de seus utilizadores ou até mesmo comprometer a imagem da organização proprietária da aplicação, tal relação de confiança passa a estar comprometida.

Algumas pesquisas foram desenvolvidas no sentido de minimizar esses problemas, com o desenvolvimento de ferramentas e técnicas que permitem a detecção e a remoção de vulnerabilidades que possam comprometer a segurança da aplicação ou a proteção contra ataques aos servidores que hospedam a aplicação.

Basicamente, as abordagens para a solução do problema podem ser divididas em duas vertentes: mudanças arquiteturais no lado dos servidores (web ou base de dados) para pré-processamento de código ([27, 10, 14]); e modificação de navegadores (extensões) para assegurar o uso da aplicação web de uma forma mais segura ([29, 12, 31]).

Enquanto a primeira abordagem, baseada em alterações no lado dos servidores, traz como principais desvantagens a difícil implantação e o inevitável incremento na latência do serviço, a segunda abordagem, baseada em alterações no lado do cliente, traz como principais desvantagens a necessidade da aprovação do utilizador para instalação de extensões e as constantes modificações necessárias nas extensões desenvolvidas para se manter a compatibilidade da mesma com novas versões dos navegadores.

O presente trabalho tem por objetivo apresentar o TRIALs (*Trusted Rich Internet Application Layers*), um framework de programação de aplicações web que permite o tratamento de um conjunto de servidores, maliciosos ou não, com o objetivo de garantir uma interação segura com uma base de dados, preservando a segurança dos dados da aplicação

web. Este framework possibilita que o navegador do utilizador de uma aplicação web possa verificar a integridade e autenticidade dos dados recebidos de um servidor web, bem como realizar a troca de servidor web de forma transparente para o utilizador, tão logo o servidor web seja considerado malicioso ou seu tempo de resposta ultrapasse um limite pré-determinado. Esta última característica permitirá o desenvolvimento de aplicações web que, não só detectem a presença de servidores web faltosos (maliciosos ou não), como também as tornem tolerante a tais tipos de faltas.

1.1 Motivação

Grande parte das aplicações web disponíveis atualmente utiliza uma arquitetura em que o código da aplicação encontra-se hospedado em um servidor web localizado em algum lugar na internet, acedido por meio de requisições HTTP (*Hypertext Transfer Protocol*) através de navegadores web nos computadores dos utilizadores. Geralmente, a aplicação web faz uso ainda de servidores de base de dados, dispostos em um ambiente com um maior nível de segurança e responsáveis pela persistência dos dados da aplicação. Neste modelo clássico, os navegadores web dos utilizadores da aplicação costumam desempenhar um pequeno papel no processamento e segurança da aplicação, limitando-se muitas vezes a realizar somente a renderização do código recebido de forma a permitir a visualização dos dados ou pequenas validações de entradas fornecidas pelo utilizador antes do envio para processamento por parte dos servidores web. A figura 1.1(a) apresenta um esquema de modelo de aplicações web clássico.

Entretanto, nos últimos anos, novas tecnologias estão despontando como uma alternativa a este modelo, onde os navegadores web têm sido responsáveis por boa parte do tratamento dos dados recebidos para sua apresentação ao utilizador. Como exemplos destas tecnologias, pode-se citar a metodologia AJAX (*Asynchronous JavaScript and XML*) [18] e os sistemas informáticos baseados em Web Services. Neste último, uma aplicação é desenvolvida de forma a receber e enviar dados em um formato padrão, permitindo que outras aplicações possam fazer uso desses dados, desde que o formato das mensagens trocadas seja mantido. Já a metodologia AJAX, permite que as aplicações web atualizem partes de suas páginas sem a necessidade de recarregá-las por completo, fazendo o uso de comunicações assíncronas com servidores e manipulações de componentes DOM (*Document Object Model*) de páginasHTML (*Hyper Text Markup Language*). Os principais padrões de formato de mensagens utilizados atualmente neste modelo são o XML (*Extensible Markup Language*) [8] e JSON (*JavaScript Object Notation*) [11]. A figura 1.1(b) apresenta um modelo de aplicações web que utiliza o AJAX e servidores web como Web Services.

O novo modelo de aplicações web com utilização do AJAX tornou-se viável devido ao significativo aumento do poder de processamento dos computadores clientes, decor-

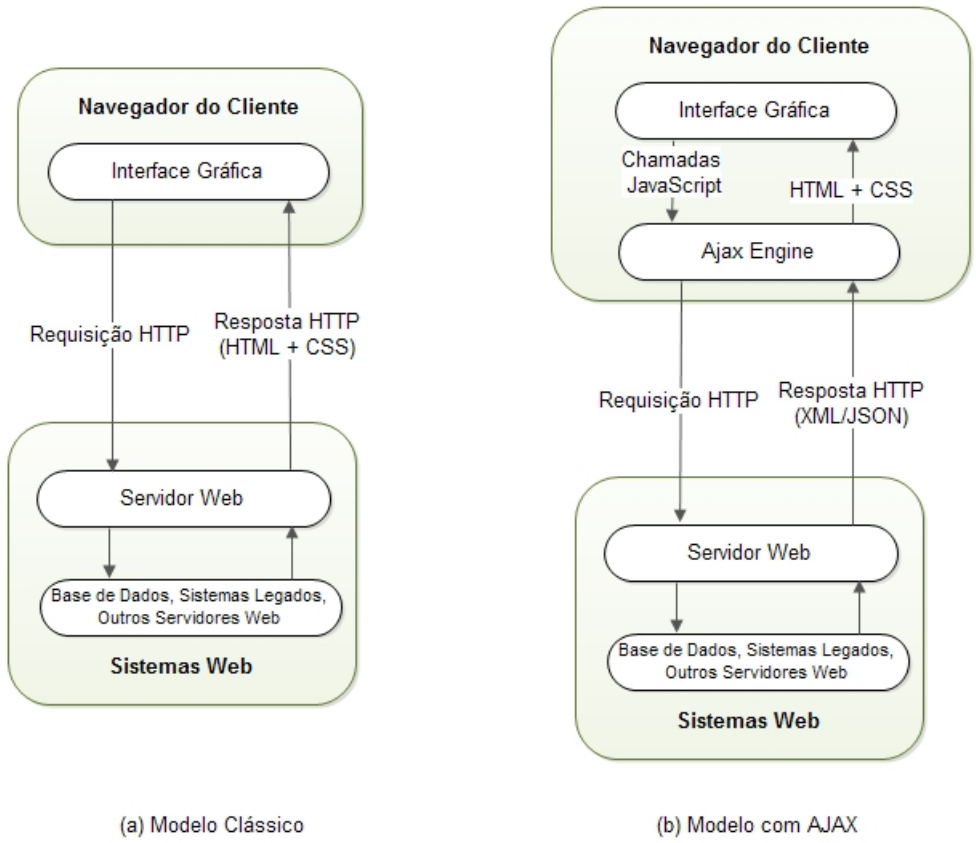


Figura 1.1: Modelo de aplicações web.

rente das melhorias constantes na velocidade dos processadores e tamanhos das memórias RAM (*Random Access Memory*) em tais computadores, em detrimento da latência na comunicação entre servidores e *desktops*. Este facto tornou possível o despontamento de aplicações web com fortes características de aplicações *desktops*, com largo uso da linguagem de programação JavaScript e utilização de navegadores web para validação e exibição de dados conforme as peculiaridades da aplicação web em questão. Este cenário mostra-se como uma alternativa ao modelo de aplicações web baseado em CGI (*Common Gateway Interface*), onde os servidores web são responsáveis por todo o processamento dos dados e contruções de páginas dinâmicas enquanto navegadores web realizam somente a renderização da página eletrônica recebida para exibição ao utilizador. Acrescenta-se a este cenário a crescente utilização de provedores de infraestrutura e serviços de tecnologia da informação para uso da tecnologia de computação em nuvem, onde os custos relacionados com réplicas de servidores como forma de garantir a segurança das aplicações podem significar aumentos no custo total de propriedade do serviço associado à aplicação, o que, por vezes, podem inviabilizar sua utilização.

Assim, torna-se mister investigar novas formas de garantir a segurança das aplicações

web, onde os navegadores web instalados nos computadores dos utilizadores, influenciados pelo aumento no seu poder de processamento, possam realizar tarefas que permitam avaliar a garantia das propriedades de segurança dos dados das aplicações web. Mais ainda, tal avaliação permitirá abordar questões referentes ao tratamento de servidores web faltosos, tanto utilizando o modelo de faltas por paragem ou o modelo de faltas arbitrárias (faltas bizantinas) [24], de forma a garantir a continuidade dos serviços mesmo na presença de tais faltas.

1.2 Objetivos

O desenvolvimento do TRIALs pretende atingir os seguintes objectivos:

- Desenvolvimento de uma API (*Application Programming Interface*) baseada em AJAX que permita a verificação da integridade e autenticidade do código recebido de um servidor web distribuiído em um ambiente de computação em nuvem. A verificação deve levar em consideração tanto o tratamento de páginas com conteúdo estático quanto das páginas com conteúdo dinâmico, com dados inseridos a partir de um conjunto de servidores de base de dados.

- Troca de servidor web de forma transparente para o utilizador da aplicação, tão logo os dados recebidos sejam considerados maliciosos ou o tempo de resposta do servidor web ultrapasse um limite pré-definido.

- Avaliação do framework com a realização de um conjunto de testes visando avaliar a fiabilidade e desempenho do mesmo.

1.3 Contribuições

Como mencionado anteriormente, as abordagens das soluções para o tratamento de servidores web não confiáveis resumem-se em duas vertentes: mudanças arquiteturais no lado dos servidores (web ou base de dados) para pré-processamento de código ([27, 10, 14]); e modificação de navegadores (extensões) para assegurar o uso da aplicação web de uma forma mais segura ([29, 12, 31]). Este trabalho apresenta como principal contribuição uma abordagem para a solução do problema com foco no uso dos navegadores web dos utilizadores para verificação da integridade e autenticidade dos dados recebidos de um servidor web, sem a necessidade de instalação de software adicional (extenções).

Uma outra contribuição importante é a possibilidade de desenvolvimento de aplicações web que sejam tolerantes a um modelo de faltas arbitrárias dos servidores web, onde os mesmos podem desviarem-se deliberadamente de seus algoritmos [2]. Em outras palavras, caso um servidor web seja comprometido por um atacante ou apresente falhas acidentais, a aplicação será capaz de detectar tal comportamento faltoso e, ainda assim,

manter o seu funcionamento correto. Isso ocorre pela funcionalidade que o framework desenvolvido oferece de realizar a troca de servidor web tão logo as propriedades de autenticidade e integridade dos dados recebidos seja violada ou um tempo limite de resposta seja atendido.

Acrescenta-se ainda como contribuição do presente trabalho a possibilidade do uso do TRIALs de forma independente da plataforma no qual os servidores web funcionam, permitindo a utilização da diversidade como forma de maximizar a tolerância a faltas proveniente de vulnerabilidades de software utilizados pelos servidores da aplicação (sistemas operativos, linguagens de programação, servidor web, sistema de gestão de base de dados, etc.).

1.4 Estrutura do documento

Este documento está organizado da seguinte forma:

- Capítulo 2 – Trabalhos Relacionados, onde serão apresentados algumas pesquisas desenvolvidas relacionadas com este projeto.
- Capítulo 3 – Framework TRIALs, onde será apresentado o trabalho desenvolvido para permitir a criação de aplicações web tolerantes a servidores web não confiáveis.
- Capítulo 4 - Implementação, onde serão apresentados os detalhes na implementação e de utilização do TRAILs, bem uma análise da segurança impplementada no mesmo.
- Capítulo 5 - Validação e Avaliação, onde serão apresentados os resultados dos testes de desempenho de uma aplicação implementada a partir do TRIALs.
- Capítulo 6 - Conclusão, onde serão apresentadas as principais conclusões tiradas do trabalho desenvolvido, bem como alguns possíveis trabalhos futuros.

Capítulo 2

Trabalho relacionado

Nesta seção serão apresentadas algumas pesquisas realizadas relacionadas com a interação de aplicações web com servidores web não confiáveis. As abordagens a este problema geralmente se resumem a duas vertentes: modificações no lado do servidor web, onde a arquitetura da aplicação é modificada de forma a resistir a ataques de servidores web comprometidos; e modificações no lado do cliente, onde software adicionais são instalados nos navegadores web dos utilizadores a fim de aumentar a segurança das aplicações web.

2.1 Soluções no lado do servidor

2.1.1 SkyProxy

Algumas pesquisas utilizaram uma abordagem baseada em mudanças arquiteturais no lado dos servidores (web e base de dados) para solucionar o problema de vulnerabilidades em aplicações web que, quando exploradas, permitem a execução de código malicioso no navegador do utilizador. Uma destas soluções é o SkyProxy [27], um analisador de conteúdo web para detecção de código malicioso baseado em execução. A figura 2.1 apresenta a arquitetura e processamento do SkyProxy. A arquitetura da solução é baseada em um proxy que intercepta as requisições HTTP do navegador do utilizador, requisita a página para o servidor web que hospeda a aplicação (figura 2.1(a)) e executa a mesma em uma máquina virtual isolada para fins de verificação da segurança do código, antes de responder à requisição do utilizador (figura 2.1(b)). Caso não sejam detectadas vulnerabilidades no código executado na máquina virtual, o proxy encaminha a resposta para o navegador web do utilizador (figura 2.1(c)). Caso a máquina virtual isolada identifique que a página solicitada possui código malicioso, a mesma não é encaminhada para o utilizador, evitando a execução deste código malicioso em seu navegador.

A solução foi avaliada pelos autores, onde, após a aplicação de algumas otimizações, apresentou uma eficácia em 100% dos casos na detecção de código malicioso e o incremento na latência do serviço de 600 milissegundos, em média. Apesar disso, os próprios

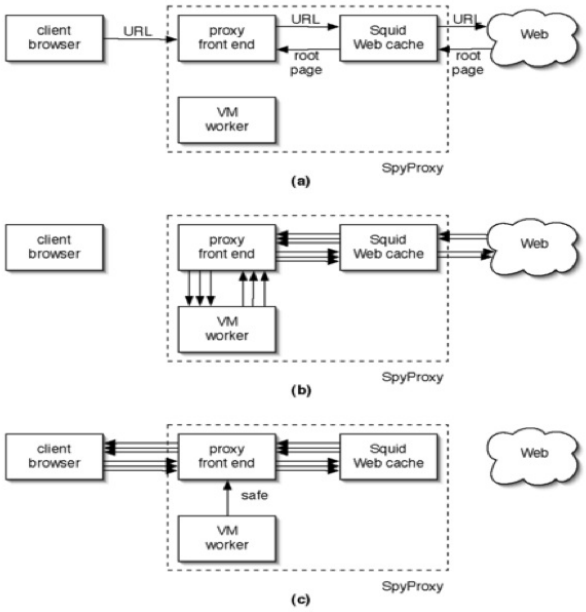


Figura 2.1: Arquitetura e processamento do SkyProxy.

autores apresentaram as seguintes limitações na solução:

- Não determinismo: conteúdos maliciosos que são executados baseados em critérios não deterministas podem não ser detectados pela solução.
- Terminação: códigos cuja terminação depende de mecanismos temporais ou entradas dos utilizadores não podem ser analisados pela solução, uma vez que a mesma utiliza interfaces dos navegadores para verificar quando uma página web foi totalmente processada.
- Diferenças entre a máquina virtual isolada e o ambiente computacional do utilizador: a diferença entre os ambientes de execução do código entre a máquina virtual isolada e o do utilizador podem fazer com que um conteúdo seja considerado inofensivo pela solução, mas comportar-se como malicioso no ambiente do utilizador devido às vulnerabilidades presentes no sistema operativo ou navegador do utilizador, não presentes na máquina virtual da solução.

2.1.2 SPARE

Um solução pesquisada que utiliza modificações na arquitetura dos servidores web para resistir à existência de servidores web maliciosos em uma aplicação web é o SPARE [14]. O SPARE é um sistema projetado para suportar especificamente sistemas baseados na web com tolerância à faltas bizantinas de maneira que os custos financeiros associados à sua arquitetura sejam compatíveis com este tipo de serviço. Utilizando a tecnologia de

virtualização de servidores disponibilizadas por soluções baseadas em hipervisores bem conhecidos no mercado e que garantem um ambiente com implementações de serviços para gestão de máquinas virtuais de forma confiável, o SPARE apresenta-se como uma solução que permite a utilização de mecanismos tolerante à faltas bizantinas em serviços web com restrições econômicas. Para tal, a solução possui uma arquitetura que requer apenas um mínimo de $f + 1$ réplicas ativas durante uma execução na ausência de falhas e f réplicas passivas que, na presença de falhas, são ativadas para garantir o correto funcionamento do sistema. Estes números são baseados na hipótese da existência de componentes confiáveis no sistema: o hipervisor e um gestor de réplicas. Além desta, são as seguintes as demais hipóteses assumidas para o sistema:

- Utilizadores interagem com o sistema somente de forma remota e utilizando mensagens do tipo *REQUEST-REPLY* através de uma rede de dados. As mensagens de interação cliente-serviço podem ser interceptadas por adversários.
- O serviço remoto pode ser modelado como uma máquina de estado determinista e a replicação de servidores é realizada utilizando técnicas de replicação de máquinas de estado conhecidas.
- O modelo de faltas adotado para as réplicas é bizantino. O sistema permanece correto mesmo na presença de f réplicas faltosas, de um total mínimo de $2f + 1$ réplicas ativas e passivas.
- O modelo de faltas adotado para o hipervisor é de paragem.
- Réplicas que falham podem ser detectadas de forma fiável em um intervalo de tempo limitado utilizando uma configuração mínima de $f + 1$ réplicas.

Supõem-se ainda a existência de um canal de comunicação fortemente interconectado e seguro, tipicamente para redes de pequena escala (LAN) e bem gerenciada. Adicionalmente, considera-se a existência de componentes confiáveis, que podem falhar somente por paragem. Um destes componentes é uma máquina virtual com altos privilégios, que tem controle total sobre o hardware e é capaz de inicializar, pausar, reiniciar e parar todas as outras máquinas virtuais hospedadas. Este componente foi denominado como gestor de réplicas. O outro componente considerado confiável e com um modelo de faltas por paragem é o próprio hipervisor, o que torna necessário a existência de no mínimo $f + 1$ máquinas físicas para o funcionamento correto do sistema. A arquitetura e processamento de requisições do SPARE estão ilustrados nas figura 2.2 e 2.3, respectivamente.

Conforme figura 2.3, o processamento básico de uma requisição de um utilizador segue os seguintes procedimentos:

- A requisição do utilizador é enviada para um dos gestores de réplica. O subcomponente do gestor de réplica responsável por este recebimento é o de comunicação de grupo;

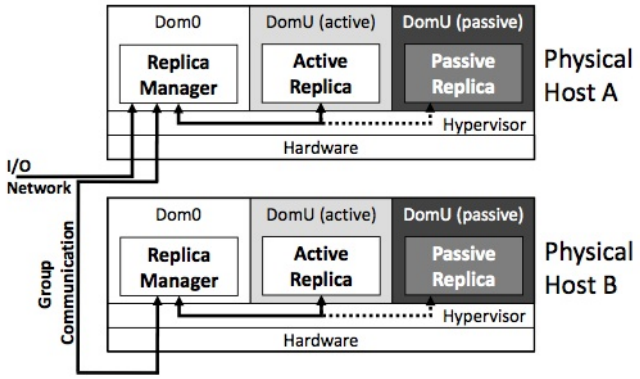


Figura 2.2: Arquitetura do SPARE.

- O componente de comunicação de grupo reencaminha a requisição para as demais réplicas, onde um mecanismo de ordenação garante uma ordem total das requisições recebidas;
- Após ordenada, a requisição é encaminhada para as réplicas ativas locais para ser processada;
- Após o processamento da requisição, as réplicas ativas presentes na máquina física que recebeu incialmente a requisição do cliente enviam para um componente de voto de resposta o resultado da requisição. As demais máquinas ativas enviam um *hash* da resposta para o componente de comunicação de grupo, que as retransmitem para as demais máquinas físicas.
- Ao receber o *hash* das respostas, o componente de comunicação de grupo encaminha as mesmas para o componente de voto de resposta.
- O componente de voto de resposta, após receber $f + 1$ sínteses que correspondem à resposta recebida pelas réplicas ativas locais, verifica que a resposta está correta e a encaminha para o utilizador.

Esta operação é realizada utilizando $f+1$ réplicas ativas, mantendo as demais réplicas em modo passivo de forma a garantir um uso eficiente de recursos na execução em um ambiente sem faltas. A detecção de faltas é baseada no mecanismo de votação de resposta utilizado e, quando ocorre, uma mensagem *HELP-ME* do gestor de réplicas é enviada para as demais réplicas utilizando o componente de comunicação de grupo, indicando o número de réplicas adicionais necessárias para responder corretamente à requisição do cliente. Todos os gestores de réplicas que não possuem réplicas passivas faltosas enviam uma mensagem *I-DO* de confirmação e uma réplica passiva é ativada para produzir mais

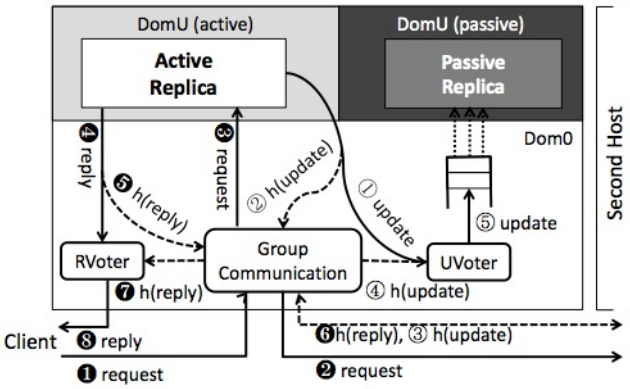


Figura 2.3: Processamento de requisições no SPARE.

uma resposta à requisição. Esta ativação é feita pelo gestor de réplicas local, que reinicializa a réplica passiva. Após a reinicialização, a réplica processa todas as requisições armazenadas em seu buffer local e processa a requisição solicitada originalmente, enviando sua resposta para o gestor de réplica que recebeu a requisição original do utilizador usando o componente de comunicação de grupo. Após o processamento da requisição, a máquina virtual pode voltar a se tornar passiva pelo gestor de réplicas ou continuar atuando como passiva. Esta decisão pode ser baseada na frequência de mensagem *HELP-ME* recebidas.

O SPARE concretiza ainda um mecanismo de recuperação proativa de forma a garantir um funcionamento correto do sistema a menos de f faltas por um período maior. Para isso, o gestor de réplica realiza um clone de uma máquina virtual passiva, reservando ela como uma nova réplica passiva. A réplica passiva original é então promovida para réplica ativa e o gestor de réplicas desliga uma das máquinas virtuais ativas anteriores. Desta forma, réplicas ativas potencialmente faltosas são eficientemente substituídas por réplicas corretas.

2.2 Soluções no lado do cliente

2.2.1 SOMA

Em contrapartida à abordagem apresentada no item anterior, algumas pesquisas optaram por solucionar o problema de execução de código malicioso nos navegadores web utilizando modificações em tais navegadores. A solução mais comum neste tipo de abordagem é o desenvolvimento de extensões de navegadores, que devem ser instaladas previamente pelos utilizadores. Uma das soluções deste tipo é o SOMA (*Same Origin Mutual Approval*) [29], que define uma política de segurança baseada em aprovação na inclusão de

conteúdo tanto pelo servidor que hospeda a página web original, quanto dos servidores que provêm conteúdos a serem inseridos na página web original. A solução é baseada em uma extensão de navegador que verifica a existência de uma relação de confiança mútua entre o servidor que hospeda a página web original e aqueles que hospedam os conteúdos incluídos nesta, utilizando ficheiros localizados em ambas as partes. A figura 2.4 apresenta um exemplo de verificação realizada pela extensão desenvolvida. A notação $A \subset B$ significa que a origem A aprova a inclusão de scripts da origem B. Similarmente, a notação $B \ni A$ significa que B aprova a inclusão de seu conteúdo na origem A. O símbolo $\not\subset$ e $\not\ni$ negam as proposições anteriores.

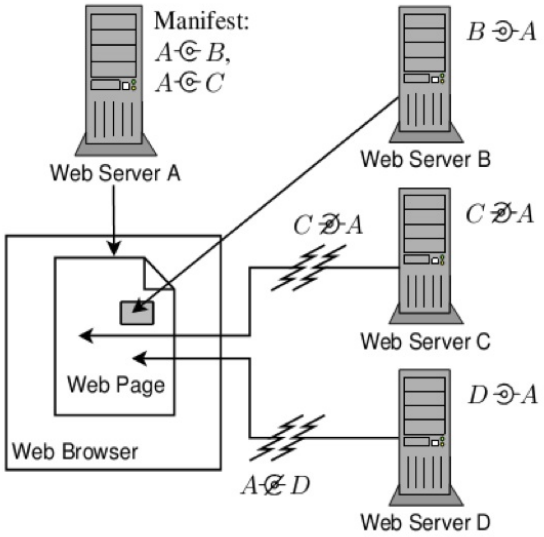


Figura 2.4: Exemplo de verificação de confiança mútua pelo SOMA.

Na figura 2.4, o servidor *A* é a hospedeiro original da página web a ser exibida e possui um manifesto que indica a aprovação da inclusão de conteúdo dos servidores *B* e *C*. Quando um navegador de um utilizador da página web recebe uma solicitação de inclusão de um conteúdo do servidor *B* na página web, realiza preliminarmente uma requisição ao servidor *B* para determinar se *B* aprova a inclusão de seu conteúdo em *A*. No exemplo, esta aprovação ocorre e o conteúdo solicitado é incluído na página web. De forma similar, o conteúdo do servidor *C* solicitado não é incluído na página web, uma vez que, mesmo que o servidor *A* aprove a inclusão de um conteúdo do servidor *C*, *C* não aprova a inclusão de seu conteúdo em páginas web oriundas de *A*. Uma outra situação presente no exemplo é a tentativa do servidor *D* de incluir conteúdo seu na página web oriunda de *A*. Esta inclusão não é realizada, uma vez que o servidor *D* não possui tal aprovação no manifesto do servidor *A*. Desta forma, o SOMA previne a inclusão de conteúdos enviados e recebidos de servidores web não aprovados e que podem ser potencialmente maliciosos.

2.2.2 PwdHash

Outro exemplo de solução que utiliza software instalado nos navegadores de utilizadores para aumentar a segurança em aplicações web é o PwdHash [31]. Trata-se de uma extensão de navegadores que, de forma transparente, produz diferentes palavras-passes para cada sítio web acessado, aumentando a segurança na autenticação da aplicação a ser utilizada.

A solução utiliza uma função *hash* para codificar as palavras-passes dos utilizadores, usando o domínio do sítio web a ser acessado como *salt*. Desta forma, ao invés de enviar para os servidores web a palavra-passe do utilizador em claro, a extensão captura-a após sua digitação, calcula o *hash* desta palavra-passe concatenada com o domínio do sítio web acessado e envia tal *hash* para o servidor web a ser acedido. As principais características na utilização desta solução é a não necessidade de realizar alterações nos servidores web de uma aplicação e a introdução de poucas modificações na forma como o utilizador utiliza a aplicação. Assim, a extensão foi projetada para agir como um intermédio transparente entre o utilizador e a aplicação, onde todos as entradas fornecidas por estes podem ser monitoradas e mantidas seguras pela extensão, antes que a aplicação tome conhecimento que o utilizador está interagindo com ela.

Os utilizadores devem notificar a extensão que eles irão digitar uma palavra-passe. Esta notificação é feita utilizando dois métodos: *Password Prefix* e *Password-Key*. No método *Password Prefix*, o utilizador deve escolher um conjunto curto de caracteres para iniciarem suas senhas. A extensão então monitora a ocorrência desta sequência para, proativamente, tomar medidas que protegem todo o campo. Na implementação, foram utilizados os caracteres @@. Uma vez detectados esses caracteres, a extensão registra todas as teclas subsequentes digitadas até que o campo de palavra-passe perca o foco. O *hash* da palavra-passe digitada concatenada com o domínio do sítio web substitui o valor do campo da palavra-passe de duas formas: logo após a perda do foco no campo ou após a submissão do formulário. Os utilizadores da extensão devem trocar as palavras-passes dos sítios web que querem proteger, utilizando o prefixo adotado para que a extensão possa enviar a nova palavra-passe codificada com o *hash* e o *salt* do domínio. Desta forma, não é necessária nehuma modificação no lado do servidor web. Se os caracteres de prefixo forem utilizados em campos que não sejam do tipo *password*, o utilizador é alertado a não digitar sua palavra-passe.

No método *Password-Key*, o utilizador escolhe uma tecla que deverá ser pressionada antes de se inserir uma palavra-passe. Na sua implementação, o PwdHash utilizou a tecla F2, não utilizada pelos navegadores Internet Explorer e Mozilla Firefox. Após pressionada esta tecla, a extensão toma as mesmas medidas proativas do método anterior. Da mesma forma, se a tecla for pressionada em campos que não sejam do tipo *password*, o utilizador é alertado a não digitar sua palavra-passe. A extensão implementa ainda uma característica adicional, composta de uma "luz" na barra de ferramentas que fica vermelha

quando a extensão encontra-se em modo texto e verde quando em modo *password*. Assim, os utilizadores podem verificar o alerta da "luz" antes de digitarem sua palavra-passe para verificarem se estão sob a proteção da extensão.

Para evitar ataques de dicionário, a solução implementou uma funcionalidade chamada de *global password*. Com esta funcionalidade, o utilizador pode configurar uma palavra-passe global, a ser utilizada também como *salt* em todas as palavras-passes a serem codificadas com a função *hash*. A desvantagem do uso desta funcionalidade é a necessidade de se configurar a mesma senha em todos os navegadores que o utilizador deseje usar. A solução ainda disponibiliza uma página web onde é possível calcular o *hash* utilizando as entradas dadas por um utilizador, sendo estas o domínio e a palavra-passe que se deseja proteger. Esta foi a solução apresentada para utilizadores que não tenham permissão de instalar extensões em seus navegadores.

2.3 Discussão

As soluções que utilizam modificação na arquitetura da aplicação no lado do servidor para realizar o tratamento de servidores web não confiáveis, apesar de se mostrarem bastantes fiáveis, trazem como desvantagem a necessidade de introdução de componentes adicionais para pré-processamento de dados. Para aplicações web que utilizam recursos computacionais distribuídos em um ambiente de computação em nuvem, a adição desses componentes é, na maioria dos casos, refletida em custos financeiros adicionais. Além disso, os estudos apresentados assumem que os componentes adicionais não se comportam de forma maliciosa, ou seja, são tidos como confiáveis, o que pode ser considerado como um modelo de ameaça fraco para um ambiente de computação em nuvem.

As soluções baseadas em mudanças no lado do cliente utilizam como principal ferramenta extensões que devem ser instaladas nos navegadores dos clientes. Estas extensões realizam o pré-processamento dos dados no lado do cliente, sem exigir mudanças arquiteturais na aplicação com adição de componentes adicionais que utilizem recursos dispostos em um ambiente de computação em nuvem. Entretanto, traz como principal desvantagem a necessidade da aceitação dos utilizadores da aplicação na instalação dessas extensões, bem como as constantes mudanças necessárias na extensão desenvolvida visando sua compatibilidade com as novas versões dos navegadores. Adiciona-se ainda as desvantagens do uso deste tipo de solução a necessidade de desenvolver tantas extensões quantas as que forem necessárias, de acordo com o tipo de navegador que os utilizadores da aplicação web possam usar.

O estudo realizado pretendeu abordar o problema do tratamento de servidores web não confiáveis de forma a evitar ao mínimo a utilização de recursos computacionais de um ambiente de computação em nuvem, evitando desta forma custos financeiros adicionais. Além disso, será apresentada uma solução que seja a mais transparente possível para

o utilizador da aplicação web, sem que seja necessária a instalação de software adicional. Para isso, o framework foi desenvolvido para programação de aplicações web baseado na tecnologia AJAX, de forma a permitir que o navegador web do utilizador possa verificar a integridade e autenticidade dos dados recebidos de um servidor web, garantindo a exibição somente de dados confiáveis. O framework é também capaz de realizar de forma transparente a troca de servidor web, tão logo as proriedades de segurança dos dados da aplicação sejam violadas ou o tempo de resposta do servidor web ultrapasse um limite de tempo pré-determinado.

Capítulo 3

Framework TRIALs

3.1 Arquitetura do Framework

O TRIALs foi desenvolvido para a criação de aplicações web cuja arquitetura é composta por três camadas: camada de apresentação, composta pela interface gráfica da aplicação web; camada de negócio, composta pela aplicação web em si, onde reside toda a lógica de negócio da aplicação hospedada em servidores web; e a camada de dados, onde os dados são armazenados em servidores de bases de dados e consultados pela camada de negócio. A figura 3.1 apresenta a arquitetura do framework desenvolvido.

Neste trabalho, a camada de apresentação, representada pelos navegadores web dos utilizadores, realiza as requisições para os servidores web mediante interação do utilizador com a aplicação web. Recebido os dados, a aplicação realizará a verificação de sua integridade e autenticidade, que, caso ocorra, são apresentados mediante manipulação de objetos DOM, ao invés de simplesmente realizar a renderização de código HTML recebido, como em um modelo de aplicações web tradicional ilustrado na figura 1.1(a). Caso os dados não sejam validados ou um tempo limite de resposta seja atingido, a camada de segurança realiza a troca de servidor web visando o correto funcionamento da aplicação web, mesmo na presença de servidores web com falhas. Essas funcionalidade foram implementadas usando a linguagem de programação JavaScript e com uso de técnicas da metodologia AJAX, encapsuladas em uma camada adicional de segurança ao modelo de aplicações web com AJAX ilustrado na figura 1.1(b).

A validação da integridade e autenticidade dos dados são feitas com o uso de funções criptográficas e de sínteses executadas nos navegadores web dos utilizadores, mediante a implementações de funções na linguagem de programação JavaScript, ou seja, sem a necessidade de instalação de software adicional nas estações de trabalho dos utilizadores.

Os servidores web são independentes de linguagems de programação, desde que sejam implementados como web services, com capacidade de receber e responder requisições com conteúdos em formatos padrões de mensagens, geralmente XML ou JSON. Isto se deve ao facto do framework ter sido desenvolvido de forma que as mensagens trocadas

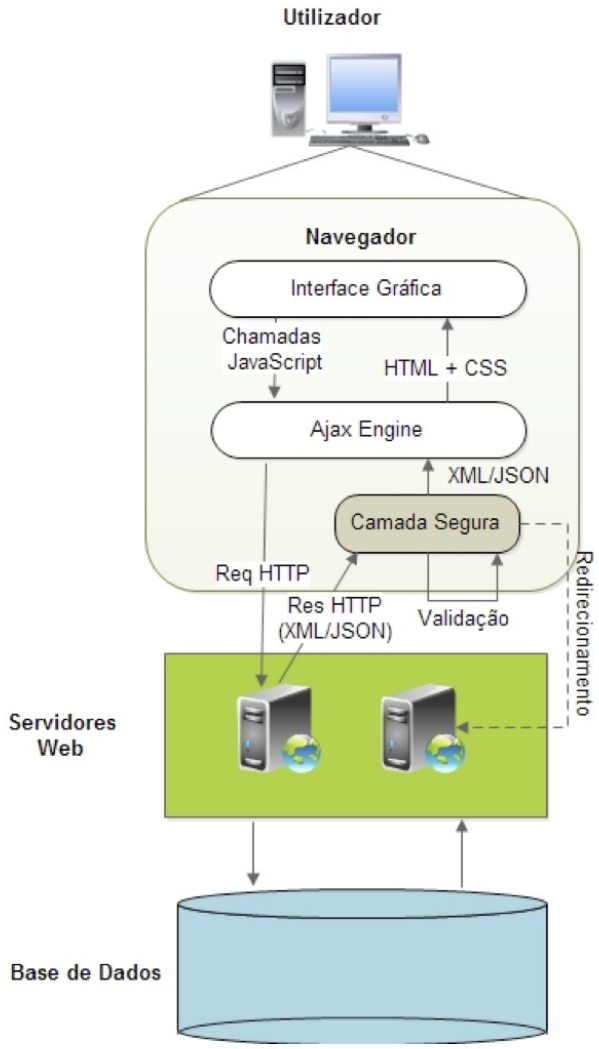


Figura 3.1: Arquitetura do TRIALs.

entre a camada de apresentação e os servidores web possuírem um formato padrão, o que vem a facilitar a função adicional de verificação da integridade e autenticidade dos dados da aplicação web.

Os servidores da base de dados são independentes de sistema de gestão de base de dados e de modelo de dados, desde que apresentem uma interface para consulta a ser utilizada pelos servidores web.

3.2 Modelo do Adversário

O TRIALs foi desenvolvido levando em consideração algumas hipóteses relacionadas com as possíveis ameaças na qual uma aplicação web desenvolvida com o seu uso possa estar exposta. A figura 3.2 apresenta o modelo do adversário, segundo as seguintes hipóteses adotadas:

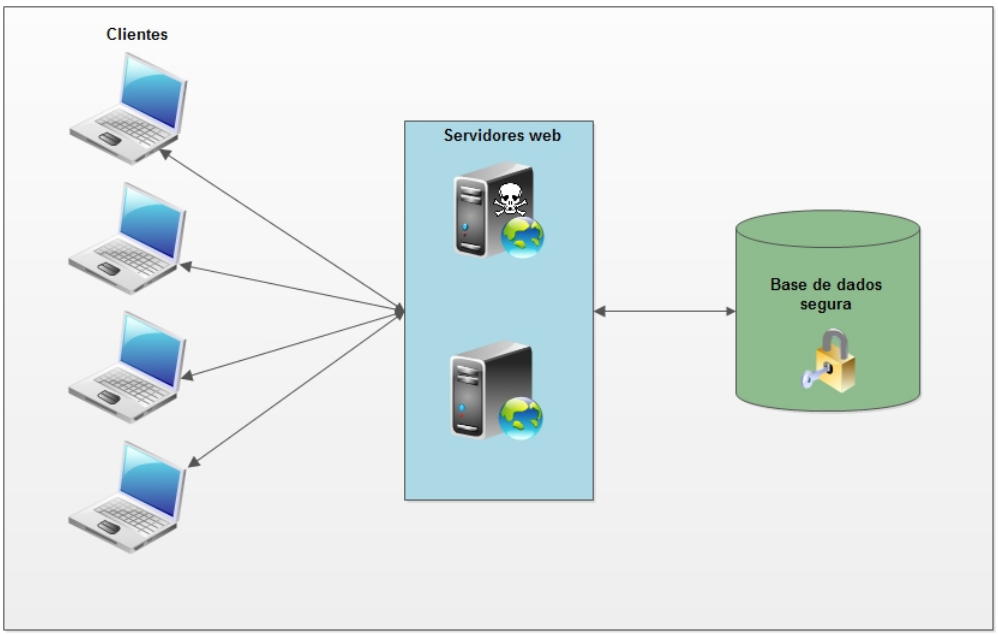


Figura 3.2: Modelo do adversário do TRIALs.

- Os servidores da aplicação web possuem um modelo de faltas bizantinas [24], ou seja, podem ser comprometidos e se comportarem de forma arbitrária, desviando-se do seu protocolo de execução, enviando respostas incorretas às requisições feitas pelos utilizadores ou atrasando intencionalmente o envio das mesmas.
- Apesar dos servidores web poderem ser comprometidos por atacantes, garante-se a existência de pelo menos um servidor correto. Ou seja, o número de servidores web disponíveis deve ser, no mínimo, igual a $f + 1$ [3].
- Os servidores de base de dados utilizam protocolos que asseguram a confiabilidade dos dados que fornecem. Em outras palavras, representam uma camada de dados tida como confiável. Isto pode ser conquistado através da instalação deste sistema em ambientes seguros ou através de técnicas de tolerância a instrusões [9].
- Um atacante ou um servidor web comprometido pode interceptar as comunicações entre os componentes da aplicação podendo alterar, atrasar, descartar ou transmitir as mensagens trocadas entre a aplicação web e a base de dados.
- Os algoritmos criptográficos (*hash*, MAC (*Message Authentication Code*), assinaturas digitais e cifras) utilizados são computacionalmente inviáveis de serem quebrados.
- O modelo temporal assumido é o parcialmente síncrono [16], uma vez que se pretende utilizar o framework em aplicações que estarão disponíveis na internet, onde a latência da comunicação entre componentes é estável a maior parte do tempo.

3.3 Problemas e Desafios

Para o desenvolvimento do TRIALs, foi necessária a realização de pesquisas sobre questões consideradas fundamentais para o seu correto funcionamento. Para algumas destas questões, foram identificadas soluções já implementadas e divulgadas na internet, como por exemplo, aquelas relacionadas com a capacidade dos navegadores web executarem algumas das funcionalidades oferecidas pelo TRIALs. Entretanto, para outras, foi necessário o desenvolvimento de soluções específicas para utilização no framework, como a definição de protocolos que permitam a leitura e escrita de dados da aplicação web por seus utilizadores de forma segura. A seguir, os problemas e desafios levantados para o desenvolvimento do TRIALs serão apresentados, juntamente com suas respectivas soluções adotadas neste trabalho.

3.3.1 Integridade do Código

O framework foi desenvolvido de forma que, uma vez que a página inicial da aplicação web seja carregada no navegador web do utilizador, todos os dados recebidos das próximas requisições HTTP são exibidos mediante manipulações dinâmicas de componentes DOM por funções JavaScript já presentes no código da aplicação recebido no primeiro acesso. Assim, uma vez que o código inicial da aplicação seja recebido, as páginas HTML subsequentes exibidas para o utilizador são criadas dinamicamente pela aplicação, utilizando os dados recebidos pelos servidores web contatados.

Foi observada, então, a necessidade de garantir que o código da aplicação recebido no primeiro acesso seja autenticado, de forma a evitar o recebimento de conteúdo malicioso por um servidor web previamente comprometido.

A solução proposta neste trabalho foi a realização da assinatura digital do código da aplicação a ser enviado no primeiro acesso, composto em sua totalidade por conteúdo estático. Esta assinatura é realizada utilizando uma chave privada do desenvolvedor da aplicação, com sua verificação sendo feita pelas estações de trabalho dos utilizadores da aplicação, utilizando o certificado digital correspondente previamente configurado em seus navegadores web. Uma pesquisa foi realizada no sentido de verificar a capacidade dos principais navegadores web atuais de realizarem a validação de assinaturas digitais. Foi constatado que o navegador Mozilla Firefox e o Internet Explorer fazem esta verificação de formas diferentes, apesar de ambos utilizarem o formato JAR (Java Archive) criado pela Sun Microsystems. No Mozilla Firefox, a verificação dos scripts assinados é feita através de uma URL específica, que aponta para um ficheiro JAR contendo os scripts/páginas e suas respectivas assinaturas [33]. Um exemplo de URL (*Uniform Resource Locator*) para verificação da assinatura de uma página é: *jar:http://www.site.com/myjar.jar!/signed.html*. No Internet Explorer, a verificação de assinaturas de scripts/páginas é feita utilizado o atributo *archive* da tag *script* ou *ob-*

ject, que deve ter como valor o ficheiro JAR contendo os scripts/páginas e suas respectivas assinaturas [25]. Um exemplo de uso deste atributo é: *script src=myscript.js archive=mysigned.jar*.

Uma outra opção seria o envio do código da aplicação juntamente com sua assinatura utilizando canais de comunicações seguro, como, por exemplo, através do PGP (*Pretty Good Privacy*)[40]. Nesta solução, o códido validado pode ser executado pelo navegador web dos utilizadores como um ficheiro local.

Neste trabalho, optou-se pela solução apresentada pelo Mozilla Firefox para fins de avaliação da aplicação web criada com o uso do framework desenvolvido, uma vez que não foi possível validar a solução apresentada pelo Internet Explorer. Foi realizado também um teste, com sucesso, com o carregamento do código da aplicação no navegador web como um ficheiro local, o que permitiria a execução de aplicações web desenvolvidas a partir do TRIALs em outros navegadores.

3.3.2 Validação de Respostas

As respostas às requisições HTTP feitas pela aplicação web devem passar por um mecanismo de validação de sua integridade e autenticidade, uma vez que servidores web comprometidos podem alterar o conteúdo dos dados fornecidos pela base de dados. No framework desenvolvido, esta validação foi implementada utilizando o mecanismo de assinaturas digitais, que permite que o criador de uma mensagem possa adicionar à mesma um código que assegure que a mensagem a ser enviada foi realmente produzida por ele e que não foi alterada após o seu envio [34].

A assinatura digital é realizada pela base de dados e anexada à mensagem a ser transmitida como resposta da requisição feita, de forma a permitir sua validação. O algoritmo escolhido para a realização das assinaturas digitais foi o RSA [30], com o uso de chaves de 512 ou 1024 bits, no formato PKCS#1, versão 2.1 [22]. O RSA é atualmente o algorítmo para realização de critografia assimétrica de blocos que tem se mostrado como o mais aceito para realização deste tipo de criptografia [34], e por este motivo foi escolhido para este trabalho.

A validação da assinatura digital é realizada pelo navegador web do utilizador, por meio de funções JavaScript presentes no framework desenvolvido. A chave pública associada à chave privada da base de dados deve ser previamente configura pelo desenvolvedor da aplicação em um ficheiro JavaScript contendo variáveis globais que representam os parâmetros de configuração da aplicação. O formato da chave pública deve ser compatível com o padrão de certificados digitais X.509 [28], versão 1 ou 3. Foi utilizada a biblioteca JavaScript *jsrsasgn* [35], desenvolvida por Kenji Urushima, para a realização da validação da assinatura digital pelo navegador web do utilizador da aplicação.

Desta forma, foi possível criar um mecanismo de garantia de integridade e autenticidade das respostas recebidas pela aplicação web utilizando o navegador web do utilizador

da aplicação, sem a necessidade de instalação de software adicional (como extensões de navegadores).

3.3.3 Integridade e Autenticidade das Escritas

A seção 3.3.2 descreveu o mecanismo utilizado no framework para garantir a integridade e autenticidade dos dados recebidos pela aplicação web utilizando assinaturas digitais. Entretanto, tal mecanismo permite apenas a autenticação da base de dados pelo utilizador da aplicação, uma vez que, feita a validação, o mesmo tem a garantia de que sua comunicação está realmente feita com a base de dados. Esta garantia é suficiente para as operações de leitura da aplicação web, mas não garante a realização de operações de escrita com integridade, já que um servidor web comprometido pode alterar os dados a serem escritos antes de encaminhá-los para a base de dados.

Sendo assim, fez-se necessário um mecanismo que permita a autenticação dos utilizadores da aplicação web junto à base de dados, bem como a criação de um mecanismo que permita à base de dados verificar a integridade e autenticidade dos dados recebidos para as operações de escrita. Em outras palavras, é necessário um mecanismo de autenticação mútua para garantir a segurança dos dados da aplicação web. A seguir, serão descritos os protocolos de autenticação e de comunicação entre a aplicação web e a base de dados implementados no framework desenvolvido.

Protocolo de autenticação

Inicialmente, será descrito o protocolo que realiza a autenticação mútua entre os utilizadores da aplicação e a base de dados, juntamente com a distribuição de uma chave de sessão, que será utilizada na verificação da integridade das mensagens no protocolo de comunicação. A figura 3.3 apresenta o protocolo de autenticação e distribuição de chave de sessão implementado no framework.

As fases iniciais do protocolo, descritas a seguir, podem ser melhor visualizadas no fluxograma ilustrado na figura 3.4. O protocolo de autenticação tem início com o fornecimento pelo utilizador de seu identificador (*ID*) e de sua palavra-passe (*p*), cujo conhecimento deve ser somente do utilizador (Fase 1). A aplicação então utiliza uma função de síntese *H* sobre a palavra-passe, produzindo uma cadeia de caracteres *P*. Em seguida, um número aleatório S_a é gerado e uma nova cadeia de caracteres *h* é criada usando a função de síntese *H* com a concatenação de *P* e S_a (Fase 2). A aplicação realiza uma requisição ao servidor web, enviando uma mensagem em formato JSON ou XML contendo o identificador *ID*, o número aleatório S_a e a cadeia de caracteres *h* (Fase 3).

A sequência de ações para a validação da autenticação pela base de dados, descrita a seguir, está ilustrada na figura 3.5. O servidor web trata a mensagem recebida, retirando os valores ID, S_a e h e enviando-os para a base de dados para que a mesma

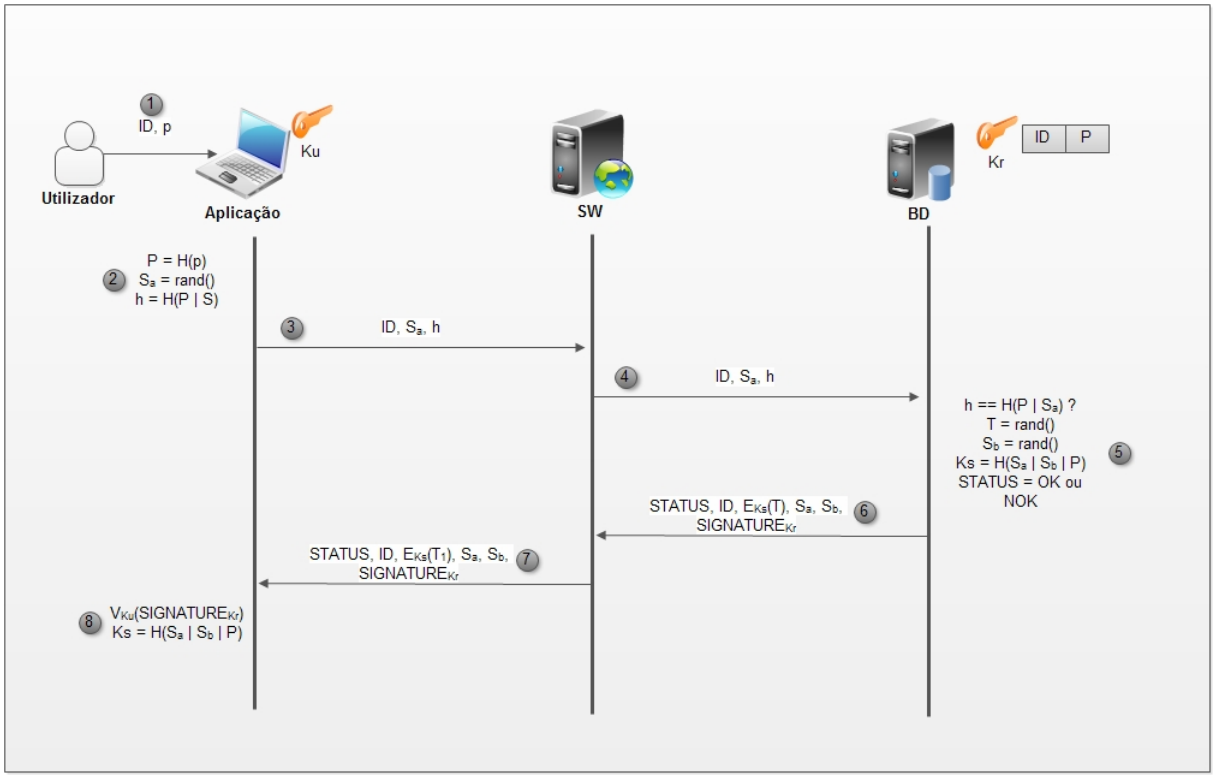


Figura 3.3: Protocolo de autenticação e distribuição de chave de sessão.

possa realizar a autenticação do utilizador (Fase 4). Esta autenticação é realizada com a verificação da igualdade entre a cadeia de caracteres h recebida, com a cadeia de caracteres calculada pela base de dados usando a concatenação do *hash* da palavra-passe do utilizador (P), armazenado nos servidores da base de dados, e a cadeia de caracteres S_a recebida ($h = H(P_{ID}|S_a)$). Realizada esta validação, inicia-se o processo de geração de uma chave de sessão que será utilizada nas comunicações futuras entre a aplicação e a base de dados. Isto é realizado utilizando a função de síntese H sobre a concatenação do número aleatório S_a recebido, um número aleatório S_b gerado pela base de dados, e o *hash* da palavra-passe do utilizador P, gerando a chave de sessão K_s = $H(S_a|S_b|P)$. Adicionalmente, outro número aleatório T é gerado e cifrado utilizando um algoritmo de cifra simétrica com a chave de sessão recém-criada K_s. Este número deve então ser armazenado na base de dados de forma associada ao identificador ID do utilizador e será utilizado nas comunicações futuras entre a aplicação e a base de dados. A base de dados cria ainda um campo adicional ($STATUS$) a ser utilizado na mensagem para informar ao utilizador sobre o sucesso ou não do processo de autenticação, cujo valor será OK ou NOK, dependendo do caso (Fase 5). A base de dados deve então enviar para o servidor web uma mensagem contendo o status da autenticação ($STATUS$), o identificador ID, o número aleatório T cifrado com a chave de sessão K_s gerada anteriormente, o número aleatório S_a, o número aleatório S_b e a assinatura de todo este conteúdo utilizando sua

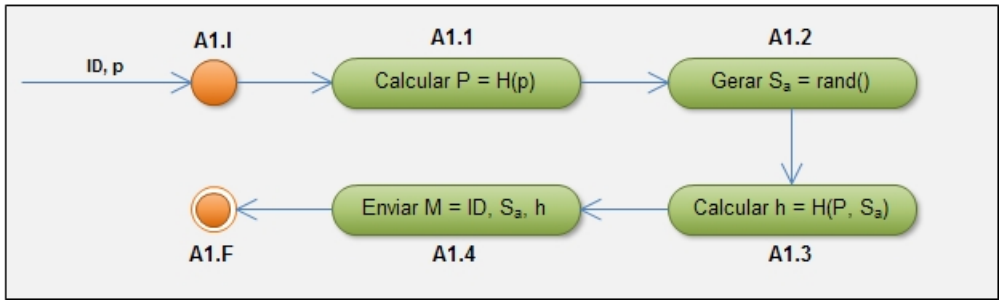


Figura 3.4: Processamento inicial da aplicação no protocolo de autenticação.

chave privada ($SIGNATURE_{Kr}$) (Fase 6). O servidor web cria uma mensagem no formato JSON ou XML contendo os valores recebidos na mensagem da base de dados e envia como resposta à aplicação (Fase 7). Caso a validação $h = H(P_{ID}|S_a)$ não ocorra, a base de dados utilizará o valor NOK para o campo $STATUS$ da mensagem de resposta, juntamente com os valores zerados dos campos $E_{Ks}(T)$ e S_a.

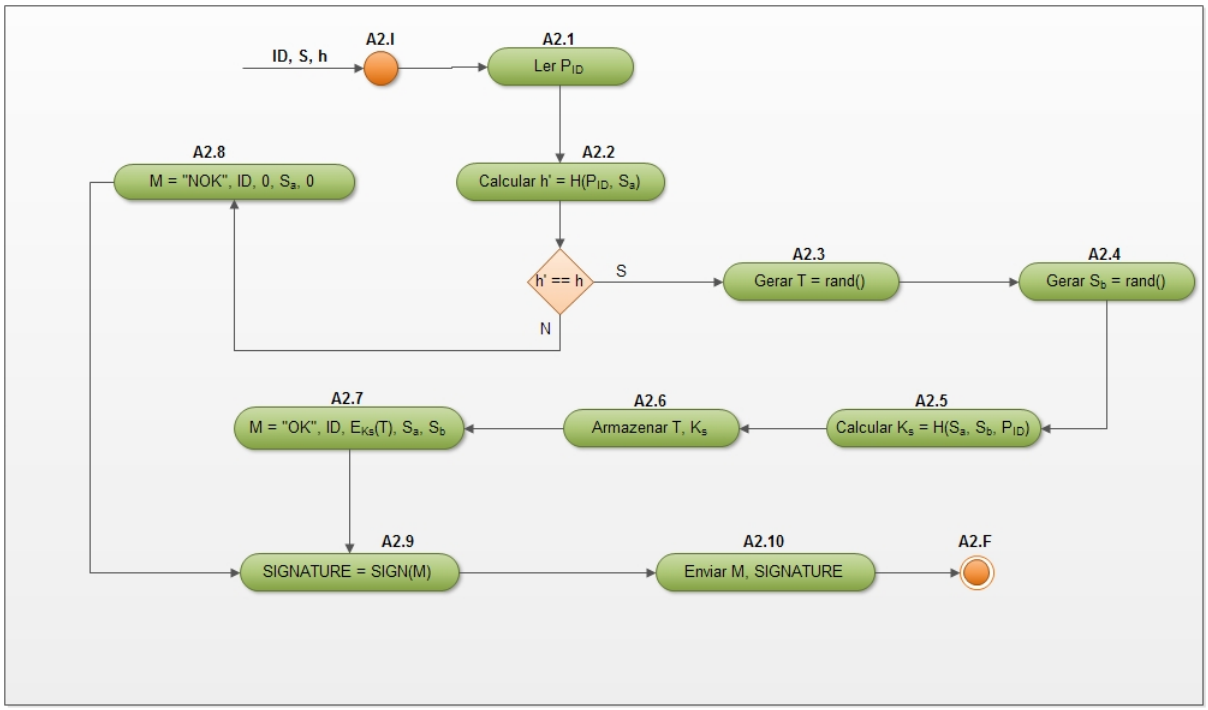


Figura 3.5: Validação da autenticação pela base de dados.

A validação da autenticação pela aplicação pode ser melhor visualizada na figura 3.6 e será descrita a seguir. A aplicação, ao receber a mensagem de resposta da base de dados, deve realizar inicialmente a validação da sua sintaxe, no que se refere ao tipo de mensagem utilizada (JSON ou XML) e a presença de todos os campos previstos no protocolo. Caso a sintaxe não seja validada, o framework realiza a troca de servidores e a

mensagem de autenticação é reenviada para o novo servidor.

Feita a verificação da sintaxe da mensagem, é realizada a validação da assinatura feita pela base de dados, utilizando a chave pública previamente configurada na aplicação. Se a assinatura da base de dados não for validada, ocorrerá a troca de servidores e o reenvio da mensagem de autenticação, como no caso descrito anteriormente.

Validada a assinatura, o campo $STATUS$ da mensagem passa por uma avaliação de seu valor, para determinar se a autenticação foi realizada pela base de dados utilizando os dados inseridos primariamente pelo utilizador da aplicação. Caso este campo possua o valor NOK, a aplicação identifica que a autenticação não foi realizada pela base de dados, realiza a troca de servidor e solicita uma nova autenticação para o utilizador. A troca de servidor web é realizada para tolerar um tipo de ataque, que será descrito na seção 3.4.

Caso o valor do campo $STATUS$ da mensagem recebida tenha valor igual a OK, serão avaliados os campos ID e S_a, para determinar se a mensagem recebida é realmente uma resposta à requisição feita. Novamente, caso tal validação não ocorra, ocorrerá a troca de servidor e um novo envio da mensagem de autenticação. Caso contrário, a aplicação deve então gerar a chave de sessão K_s, utilizando a função de síntese H na concatenação entre os números aleatórios S_a e S_b recebidos e a cadeia de caracteres P gerada anteriormente a partir da palavra-passe fornecida pelo utilizador. O número T pode então ser decifrado pelo mesmo algoritmo de cifra simétrica utilizado pela base de dados utilizando a chave K_s gerada (Fase 8). Ao final deste processo, a autenticação mútua estará realizada e uma chave de sessão é estabelecida para as comunicações futuras entre a aplicação e a base de dados.

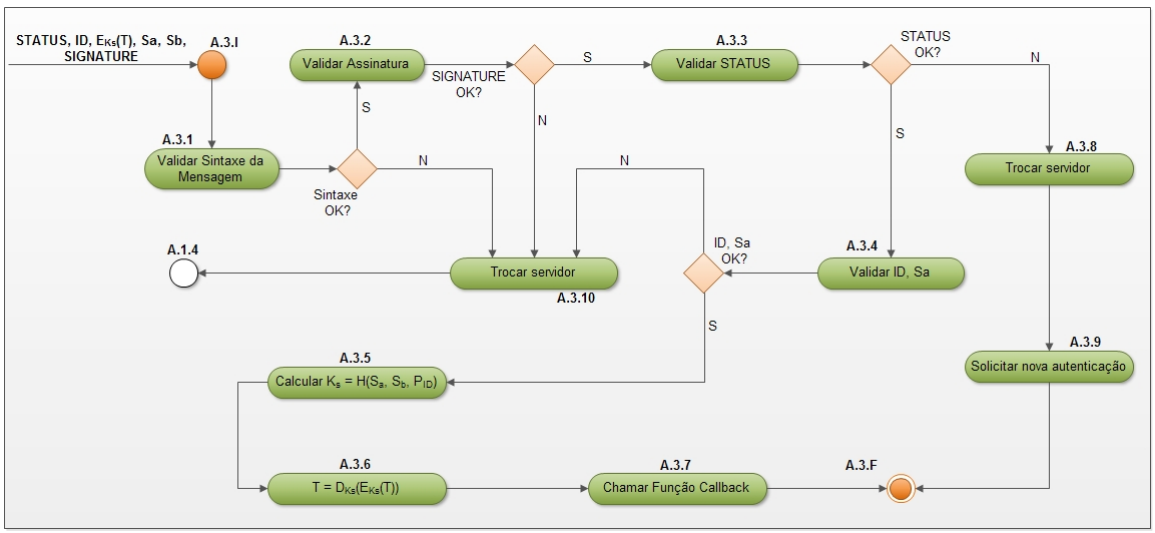


Figura 3.6: Validação da autenticação pela aplicação.

Protocolo de comunicação

Estabelecida a autenticação mútua entre o utilizador da aplicação web e a base de dados, juntamente com a chave de sessão, fez-se necessária a criação de um protocolo que permita a interação entre estes de forma segura, garantindo que as mensagens trocadas entre os mesmos, caso sejam alteradas, sejam descartadas pelo respectivo receptor. Em suma, é necessário garantir a integridade das mensagens. Além disso, identificou-se a necessidade da criação de um mecanismo adicional que evite que mensagens já transmitidas sejam processadas novamente pela base de dados ou reencaminhadas para a aplicação como resposta a uma operação subsequente. Tais necessidades devem-se ao facto da capacidade de um servidor web alterar as mensagens trocadas entre a aplicação web e a base de dados ou retransmití-las em outro contexto.

Para a garantia da integridade dos dados trocados entre a aplicação e a base de dados e da identificação de mensagens retransmitidas, o protocolo de comunicação apresentado na figura 3.7 foi implementado.

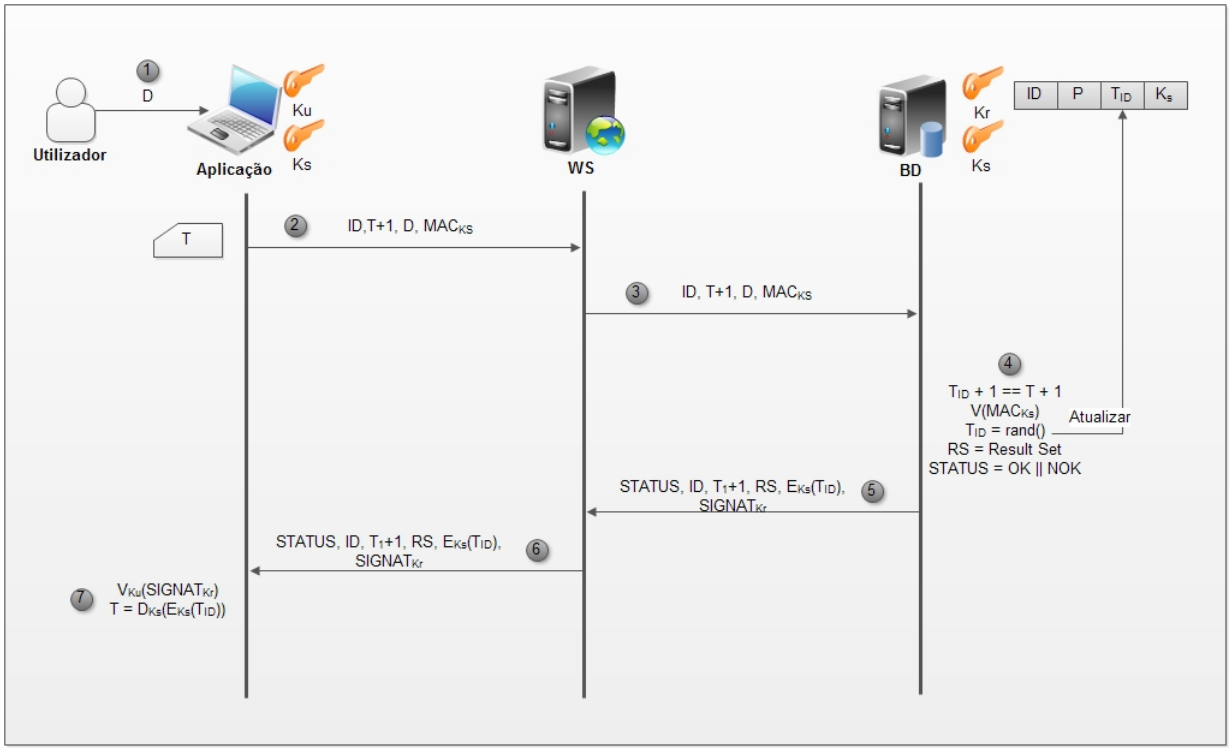


Figura 3.7: Protocolo de comunicação aplicação-base de dados com garantia de integridade.

O protocolo de comunicação utilizou o *Message Authentication Code* (MAC) como mecanismo de verificação da integridade e autenticidade das mensagens, utilizando a chave de sessão estabelecida no protocolo descrito anteriormente. Este mecanismo permite que o receptor de uma mensagem possa verificar que os dados recebidos são exa-

tamente os mesmos que foram enviados (ou seja, não sofreram modificações, inserções ou supressões) e que tal mensagem foi gerada pelo emissor correspondente a sua identificação [34] (p. 363). Adicionalmente, foi implementado um mecanismos de *tokens*, representado no protocolo como o número aleatório T, a ser utilizado como prevenção no processamento de mensagens retransmitidas por um servidor web comprometido. Este número será transmitido utilizando o algoritmo de cifras simétricas AES [13] com a chave de sessão estabelecida no protocolo anterior.

As fases iniciais do protocolo de comunicação podem ser visualizadas na figura 3.8. O protocolo tem início com a inserção por parte do utilizador dos dados que ele deseja ler ou escrever na base de dados, que pode ser feito, por exemplo, mediante formulários HTML (Fase 1). Inseridos tais dados, a aplicação realiza uma requisição para o servidor web utilizando uma mensagem no formato JSON ou XML contendo o identificador ID do utilizador, o valor $T + 1$, os dados inseridos pelo utilizador (D) e uma cadeia de caracteres que representa o MAC de todos os dados anteriores, gerado usando também a chave de sessão K_s. O servidor web, ao receber tal requisição, trata a mensagem recebida de forma a retirar todos os valores presentes e encaminhá-los para a base de dados para que a mesma possa processar a requisição da aplicação (Fase 3).

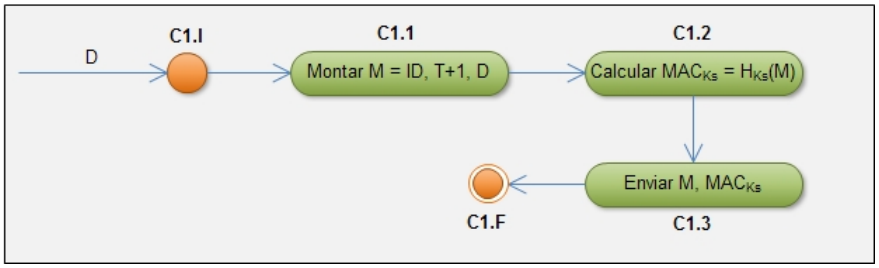


Figura 3.8: Início do protocolo de comunicação pela aplicação web.

A sequência das fases para o processamento da mensagem de requisição da aplicação pela base de dados pode ser melhor visualizada na 3.9 e será descrita a seguir. A base de dados inicialmente realiza a verificação da integridade dos dados através da cadeia de caracteres MAC enviada pela aplicação. Para isso, recupera de seus registos os valores de K_s e T associados ao ***ID*** presente na mensagem, calcula MAC' utilizando a mesma função síntese utilizada pela aplicação web nos campos ID, $T + 1$, D e a chave K_s recuperada. O valor de MAC' é então comparado com o campo MAC recebido na mensagem. Caso os valores sejam idênticos, a integridade e autenticidade da mensagem recebida é garantida e o protocolo tem prosseguimento. Caso contrário, a base de dados atribui o valor NOK para o campo $STATUS$, assina a mensagem a ser enviada como resposta contendo os campos $STATUS$, ID e $T + 1$ utilizando a sua chave privada K_re envia tal mensagem para o servidor web, a fim de que o mesmo retransmita-a para a aplicação web.

Validada a integridade e autenticidade da mensagem recebida, a base de dados com-

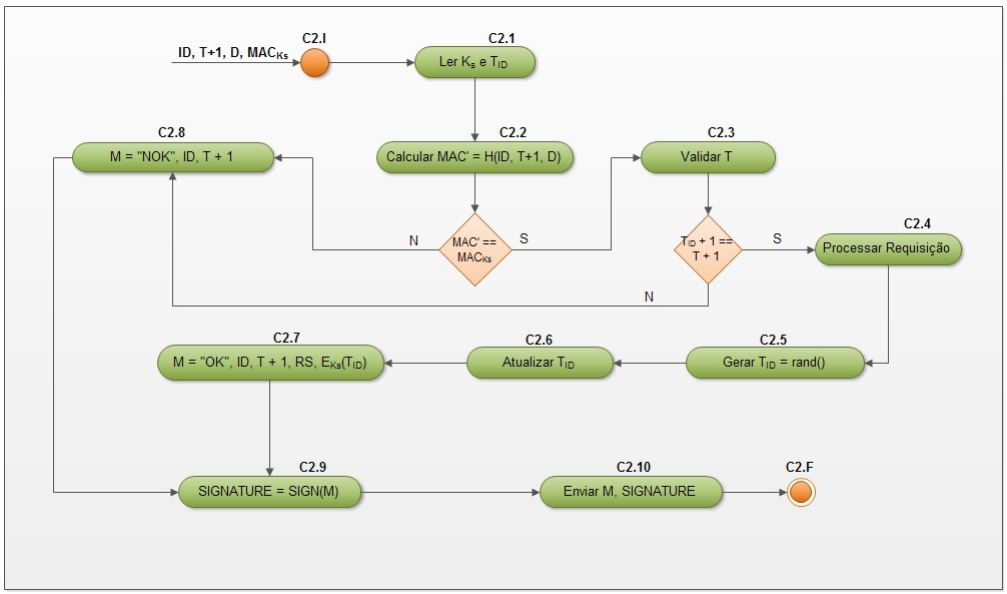


Figura 3.9: Processamento da mensagem de requisição da aplicação pela base de dados.

para o campo $T + 1$ com o valor de T_{ID} recuperado anteriormente. Caso a adição seja verificada, a base de dados processa a requisição do utilizador usando os dados recebidos, gerando um resultado RS, bem como o $STATUS$ da operação, que pode receber o valor OK ou NOK, dependendo do sucesso ou não do processamento da requisição. Adicionalmente, gera um novo número aleatório T_{ID}, substituindo-o pelo valor armazenado anteriormente (Fase 4). Este número será utilizado na próxima requisição da aplicação. A base de dados então envia para o servidor web uma mensagem contendo o status da operação realizada ($STATUS$), o identificador ID do utilizador, o valor $T + 1$, o resultado da requisição RS, o novo valor de T_{ID} cifrado com um algoritmo de cifra simétrica e com a chave K_s e a assinatura da mensagem usando sua chave privada K_r (Fase 5). Esta mensagem é recebida pelo servidor web, que a utiliza para montar a mensagem no formato JSON ou XML, que representará a resposta da requisição enviada pela aplicação (Fase 6). Opcionalmente, pode-se utilizar nesta mensagem o MAC da mensagem a ser enviada como resposta utilizando a chave de sessão estabelecida no protocolo anterior, em substituição a assinatura digital.

O recebimento da resposta da base de dados pela aplicação web está ilustrado na figura 3.10 e será descrito a seguir. A aplicação web, ao receber a resposta da sua requisição, realizará a validação da sintaxe da mensagem, a exemplo do ocorrido no recebimento da mensagem no protocolo de autenticação. Novamente, caso a sintaxe não seja validada, o framework realiza a troca de servidor e uma nova requisição é enviada para o novo servidor. Validada a sintaxe da mensagem, a aplicação valida a assinatura digital da base de dados para verificar sua integridade e autenticidade. Caso não ocorra tal validação, é realizada a troca de servidor e uma nova requisição é enviada. Ocorrendo a validação

da assinatura, a aplicação verifica se o campo $STATUS$ contém o valor OK, de forma a verificar se a requisição feita foi processada pela base de dados de forma correta. Caso esta verificação não ocorra, é realizada a troca de servidor e solicitado ao utilizador uma nova autenticação. A necessidade dessa nova autenticação, ao invés da retransmissão da mensagem como feito anteriormente, será descrita na seção 3.4. Feita a validação do campo $STATUS$, a aplicação verifica os campos ID e $T + 1$, visando identificar se a mensagem recebida trata-se de uma resposta à última requisição feita. Caso negativo, os procedimentos de troca de servidor e reenvio da requisição é realizado. Validados estes campos, a aplicação web pode utilizar os dados contidos no campo RS para exibição ao utilizador por meio de manipulação de objetos DOM. A aplicação deve também decifrar o número aleatório T utilizando o mesmo algorítmo de cifra simétrica utilizado pela base de dados e a chave de sessão K_s, atualizando seus registos para permitir o seu uso na próxima requisição (Fase 7).

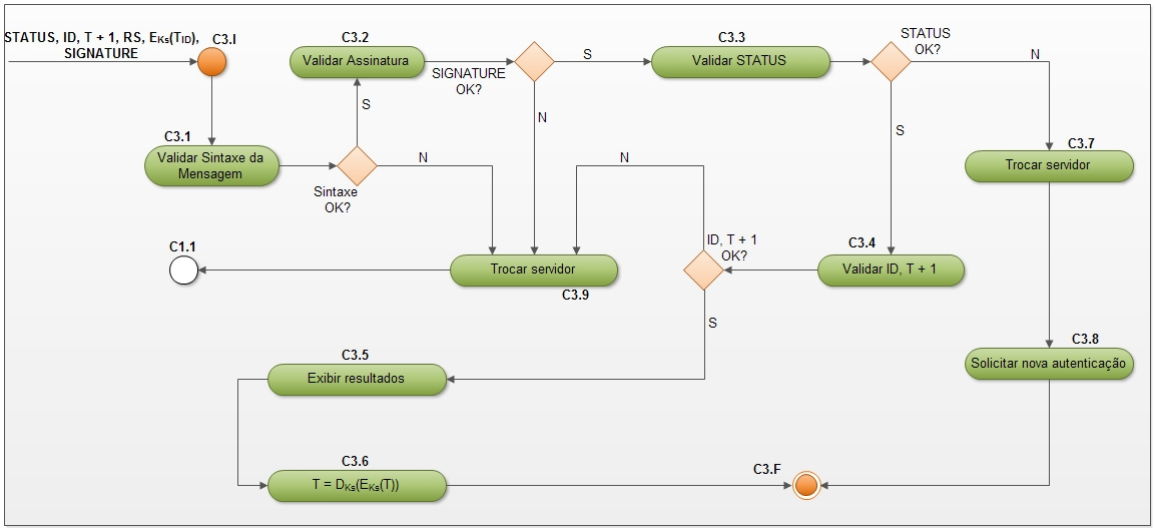


Figura 3.10: Recebimento da mensagem resposta da base de dados pela aplicação web.

3.3.4 Troca de Servidores Web

O TRIALs foi desenvolvido de forma a permitir que uma aplicação web seja tolerante a faltas, sejam elas maliciosas ou não. Em outras palavras, a aplicação deve continuar o seu funcionamento correto mesmo após a ocorrência de falhas em um dos seus componentes. Em uma arquitetura de aplicação web em três camadas, geralmente, a consistência dos dados é impelementada com servidores de bases de dados que armazenam informações essenciais para a organização proprietária da aplicação web. Com isso, geralmente, esta camada é instalada utilizando mecanismos que garantam um ambiente o mais seguro possível, como por exemplo *firewalls*, sistemas de detecção e prevenção de

intrusões, correlacionadores de eventos, segurança física, dentre outras. Com estes mecanismos, é possível criar um ambiente seguro e que seja protegido de intrusões de forma a garantir a existência de uma base de dados confiável e permitir o funcionamento de uma aplicação web desenvolvida com o TRIALs.

Uma outra opção seria o uso de técnicas de tolerância a intrusões, onde faltas intencionais e maliciosas são permitidas, porém toleradas, uma vez que o sistema utiliza mecanismos que previne tais faltas de gerar um comportamento indesejado no sistema ([38]). Pode-se citar como alguns exemplos desta técnica: a utilização de componentes confiáveis, arquitetura híbrida, criptografia de limiar, replicação de máquinas de estado, dentre outros. Diversas pesquisas vem sendo feitas nas últimas décadas sobre tais mecanismos, com apresentação de soluções práticas de seu uso ([41, 7, 37, 42]). O uso de tais mecanismos permitiria a criação de uma base de dados confiável necessário para o funcionamento de uma aplicação web desenvolvida com o TRIALs, devendo os mesmos acrescentar em sua arquitetura a camada proposta pelo TRIALs para a realização das assinaturas digitais das mensagens.

Para a tolerância de servidores web faltosos, foi implementado no framewok uma funcionalidade de troca de servidor web, tão logo seja verificado pela aplicação a não integridade e/ou autenticidade de uma mensagem recebida, ou que um limite de tempo pré-definido seja alcançado na espera de uma resposta de uma requisição enviada.

A política de origem comum de navegadores web previne que um documento ou script carregado de uma origem recupere ou altere propriedades de um documento de outra origem [36]. Desta forma, a comunicação entre domínios diferentes em uma mesma página é restringida. Entretanto, para permitir a tolerância a faltas em servidores web em aplicações web desenvolvidas a partir do TRIALs, tal comunicação entre domínios diferentes é necessária. Para permitir a comunicação de uma mesma aplicação web com servidores com domínios diferente, o framework utilizou uma técnica definida como CORS (*Cross-Origin Resource Sharing*) [36]. Esta técnica consiste de uma simples modificação no cabeçalho da comunicação HTTP entre o servidor e o cliente. Assim, o uso do atributo *Access-Control-Allow-Origin* no cabeçalho HTTP da resposta do servidor, contendo como valor uma relação dos domínios no qual o mesmo aceita o compartilhamento do recurso permitirá a troca de dados entre estes domínios. Para exemplificar, suponha um recurso presente em *http://dominio2.com/alomundo*, cujo conteúdo seja a string *Alô Mundo*. Para permitir que o domínio *http://dominio1.com* seja capaz de acessar tal recurso, a resposta HTTP deste recurso deve ser configurada para possuir em seu cabeçalho, além dos obrigatórios, o atributo *Access-Control-Allow-Origin* com o valor *http://dominio1.com*, conforme listagem 3.1.

```
1 Access-Control-Allow-Origin: http://dominio1.com
2 Alo Mundo!
```

Listagem 3.1: Cabeçalho HTTP de um servidor web utilizando CORS

Desta forma, usando o objeto XMLHttpRequest [4] da API ECMAScript [21], uma aplicação cliente oriunda de *http://dominio1.com* pode acessar o recurso disponível em *http://dominio2.com/alomundo*, conforme listagem 3.2.

```
1 var client = new XMLHttpRequest();
2 client.open("GET", "http://dominio2.com/alomundo")
3 client.onreadystatechange = function() { /* do something */ }
4 client.send()
```

Listagem 3.2: Exemplo de Requisição AJAX para acesso a diferentes domínios

A figura 3.11 apresenta um exemplo do uso da técnica CORS para a comunicação de uma aplicações web com servidores de diferentes domínios.

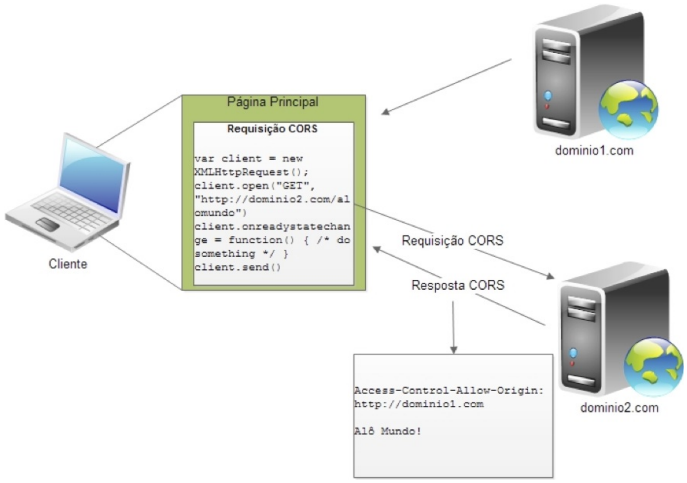


Figura 3.11: Exemplo de uso de CORS

Para o desenvolvimento de uma aplicação web com o framework desenvolvido neste trabalho, os servidores web a serem utilizados devem então estar configurados de forma a permitir o funcionamento da técnica CORS, ou seja, todas as respostas às requisições feitas devem conter em seu cabeçalho HTTP o atributo *Access-Control-Allow-Origin* com valor igual a lista dos domínios dos demais servidores web utilizados.

A funcionalidade de troca de servidores web foi implementada no framework utilizando uma lista contendo o endereço de todos os servidores web disponíveis, que deve estar presente em um ficheiro JavaScript que contém os parâmetros de configuração da aplicação. Os endereços dos servidores web vão sendo utilizados na ordem em que são inseridos na lista, até que um servidor web correto seja contatado. Cabe ressaltar que o número de servidores web presente na lista deve ser, no mínimo, igual ao número de faltas que se pretende tolerar, somado mais uma unidade. Este número garante a existência de pelo menos um servidor web correto que, inevitavelmente, será contatado no caso de ocorrência de falhas.

Nota-se com esta implementação que a funcionalidade de troca de servidor web ocorre de forma transparente ao utilizador da aplicação web, tanto no que se refere a ocorrência

de falhas, quanto à localização dos servidores web contatados.

3.4 Análise da Segurança

Os ataques efetuados por um servidor web comprometido podem basicamente ser divididos em cinco tipos:

1. Alteração do código fonte da aplicação enviado no primeiro acesso.
2. Atrasar ou deixar de encaminhar mensagens.
3. Alterar a sintaxe das mensagens, excluindo ou alterando os nomes dos campos das mensagens
4. Alteração os campos das mensagens.
5. *Replay* de mensagens enviadas em operações anteriores.

Ataque 1 No primeiro tipo de ataque, um servidor web foi comprometido antes do primeiro acesso e o código fonte da aplicação foi modificado pelo atacante. Para evitar que o código malicioso seja executado no navegador web do utilizador, o framework utiliza um mecanismo de assinatura digital descrito na seção 3.3.1, utilizando um par de chaves pública-privada entre o desenvolvedor e o utilizador da aplicação. Uma outra opção para a resistência a este ataque é o envio do código da aplicação utilizando canais seguros, como por exemplo, o PGP [40].

Ataque 2 O segundo tipo de ataque (atraso ou omissão) é tolerado pelo framework com o uso do método *setTimeout* do objeto *Window* da linguagem JavaScript, que permite chamar uma função ou avaliar uma expressão após um número de milisegundos especificado. Com a utilização deste método no framework desenvolvido, foi possível configurar um limite de tempo no qual o navegador web deve esperar pela resposta a uma requisição, limite este que, uma vez atingido, causa a troca de servidor web de forma transparente para o utilizador.

Ataque 3 Os ataques do terceiro tipo referem-se a capacidade do servidor web comprometido alterar o padrão das mensagens trocadas entre a aplicação web e a base de dados. Para tolerar este ataque, o framework faz uma validação da sintaxe das mensagens recebidas, no que se refere ao seu formato (JSON ou XML) e a presença de todos os campos necessários para o funcionamento correto da aplicação. Caso esta validação não seja feita, o framework realiza a troca de servidor web, tolerando o ataque em questão. A tolerância a este tipo de ataque pode ser melhor visualizada pelos caminhos $A3.I \rightarrow$

$A3.1 \rightarrow A3.10 \rightarrow A1.4$ (figura 3.6) e $C3.I \rightarrow C3.1 \rightarrow C3.9 \rightarrow C1.1$ (figura 3.10) dos protocolos de autenticação e de comunicação, respectivamente.

Ataque 4 Os ataques do quarto tipo são, em sua grande maioria, tolerados pelos mecanismos de integridade implementados pelo framework: *Message Authentication Code* (MAC) usando a chave de sessão estabelecida após a autenticação mútua (K_s), para o caso das mensagens enviadas pela aplicação em execução no navegador web do utilizador; e assinaturas digitais com o uso da chave privada da base de dados e a correspondente chave pública pré-configurada no código da aplicação, para o caso das mensagens enviadas pela base de dados. Alterações nas mensagens de resposta da base de dados no protocolo de autenticação ($ID, EKs(T), S_a, S_b, SIGNATURE$) e no protocolo de comunicação ($STATUS, ID, T+1, RS, EKs(T_{ID}, SIGNATURE)$, são identificadas com a validação da assinatura digital feita pela base de dados e toleradas com a troca de servidor web e realização de uma nova requisição. Os caminhos do fluxograma seguido neste ataque serão, então, $A3.I \rightarrow A3.2 \rightarrow A3.10 \rightarrow A1.4 \rightarrow A1.F$ (figuras 3.6 e 3.4), para o caso do protocolo de autenticação e $C3.I \rightarrow C3.1 \rightarrow C3.2 \rightarrow C3.9 \rightarrow C1.1 \rightarrow C1.2 \rightarrow C1.3 \rightarrow C1.F$ (figuras 3.10 e 3.8), para o caso do protocolo de comunicação. De forma semelhante, alterações na mensagem de envio no protocolo de comunicação ($ID, T+1, D, MAC_{Ks}$) são identificadas com a validação pela base de dados do MAC da mensagem feito com a chave de sessão K_s. Esta não validação acarretará em uma mensagem de resposta contendo o campo $STATUS$ com valor NOK, que será identificada pela aplicação e tolerada pela troca de servidor web e solicitação de nova autenticação. O caminho do fluxograma seguido neste caso é $C2.I \rightarrow C2.1 \rightarrow C2.2 \rightarrow C2.7 \rightarrow C2.8 \rightarrow C2.9 \rightarrow C2.F \rightarrow C3.I \rightarrow C3.1 \rightarrow C3.2 \rightarrow C3.7 \rightarrow C3.8 \rightarrow C3.F$ (figura 3.9 e figura 3.10).

A exceção no uso de mecanismos de integridade para tolerar ataques de modificação de mensagens é feita para a primeira mensagem enviada para a base de dados na fase de autenticação (ID, S_a, h), justamente por ser a única mensagem do protocolo que não possui um mecanismo de integridade adicionado. Entretanto, caso algum campo desta mensagem seja alterado por um servidor web comprometido, a autenticação junto a base de dados não será realizada, pois a verificação $h == H(P|S_a)$ não será válida e a base de dados responderá a requisição com o campo $STATUS$ da mensagem como NOK, indicando para a aplicação web que não foi possível realizar a autenticação do utilizador. Desta forma, o ataque acarretará na não disponibilidade da aplicação web para um utilizador credenciado. Para este caso, a tolerância do ataque é alcançado pela realização da troca de servidor web, sempre que uma mensagem de autenticação contendo o campo $STATUS$ com valor NOK. Como esta mensagem também será recebida no caso de ser contatado um servidor web correto, mas a combinação ID - palavra-passe inserida pelo utilizador ser inválida, esta troca será realizada tantas vezes quantas forem necessárias

até que um servidor correto seja contatado e o identificador ID e a palavra-passe sejam fornecidos pelo utilizador de maneira correta. Para este ataque, a tolerância pode ser visualizada pelo caminho $A2.I \rightarrow A2.1 \rightarrow A2.2 \rightarrow A2.8 \rightarrow A2.9 \rightarrow A2.10 \rightarrow A2.F \rightarrow A3.I \rightarrow A3.2 \rightarrow A3.3 \rightarrow A3.8 \rightarrow A3.9A3.F$ (figuras 3.5 e 3.6).

Ataque 5 Os ataques do quinto tipo, na qual as mensagens são retransmitidas, seja para a base de dados ou para os utilizadores da aplicação, é também conhecido como ataque de *replay*. Este tipo de ataque é caracterizado basicamente pelo armazenamento de mensagens ou partes dela, retransmitindo-as em outro contexto. As mensagens podem ser redirecionadas para outros destinatários diferentes daqueles originariamente pretendidos, ou podem ser repetidas em diferentes instâncias do protocolo ou passos de transmissão [5]. Assim, um servidor web comprometido pode, por exemplo, armazenar a mensagem enviada pela aplicação web para o início da autenticação mútua (ID, S_a, h), e retransmití-la para a base de dados com o objetivo de personificar um determinado utilizador. Caso isto ocorra, a base de dados realizará a autenticação do pseuso-utilizador, estabelecerá uma chave de sessão (K_s) para as comunicações futuras e encaminhará a mensagem prevista no protocolo ($STATUS, ID, E_{Ks}(T), S_a, S_b, SIGNATURE$) para o servidor web comprometido. Entretanto, como a chave de sessão é gerada utilizando o segredo P, que é do conhecimento somente da base de dados e do utilizador, não será possível ao servidor web comprometido calcular a chave de sessão e, consequentemente, decifrar o número aleatório T gerado, para permitir a realização da próxima requisição. Um problema adicional neste tipo de ataque é a não possibilidade de realização de requisições futuras pelo utilizador com o ID presente na mensagem retransmitida e autenticado anteriormente a tal retransmissão. Isto ocorre pelo facto da retransmissão da mensagem ocasionar o estabelecimento de uma nova chave de sessão K_s, fazendo com que a base de dados não valide a integridade da mensagem enviada pela aplicação para a realização da operação. A consequência desta não validação é o uso do valor NOK no campo $STATUS$ da mensagem de resposta da base de dados, o que acarretará na troca de servidor web e solicitação de nova autenticação. Assim, este ataque acaba por ser ineficiente também devido a um mecanismo de integridade: o MAC presente na mensagem do protocolo de comunicação enviada pela aplicação web ($ID, T + 1, D, MAC_{Ks}$), cuja defesa pode ser representada no caminho $A2.I \rightarrow A2.1 \rightarrow A2.2 \rightarrow A2.8 \rightarrow A2.9 \rightarrow A2.10 \rightarrow A2.F \rightarrow A3.I \rightarrow A3.2 \rightarrow A3.3 \rightarrow A3.8 \rightarrow A3.9A3.F$ (figuras 3.5 e 3.6).

Um servidor web comprometido pode ainda retransmitir a mensagem $STATUS$, ID, $E_{Ks}(T)$, S_a, S_b, $SIGNATURE_{Ks}$ para a aplicação web como resposta a uma mensagem para realização da autenticação mútua, com o objetivo de personificar a base de dados. Nesta caso, a tolerância do framework para este ataque é através da utilização do número aleatório S_a, gerado em cada uma das operações de autenticação pela aplicação web em execução no navegador web do utilizador. Ao receber a mensagem de resposta

da autenticação mútua, a aplicação web fará inicialmente a validação do número aleatório S_a, comparando-o com o que foi enviado na mensagem inicial (ID, S_a, h). Tal validação não será efetuada, para o caso da retransmissão de uma mensagem em questão e ocorrerá a troca de servidor web e o reenvio da mensagem da requisição, facilmente visualizada pelo caminho $A3.I \rightarrow A3.1 \rightarrow A3.2 \rightarrow A3.3 \rightarrow A3.4 \rightarrow A3.10 \rightarrow A1.4 \rightarrow A1.F$ (figuras 3.6 e 3.4).

Um outro ataque possível é a retransmissão da mensagem $\boldsymbol{ID}, T+1, D, MAC_{Ks}$ por um servidor web comprometido, com o objetivo de personificar o utilizador da aplicação web. Ao receber tal mensagem, o protocolo implementado determina a verificação do campo $T+1$ por parte da base de dados, com relação ao número T relacionado com o identificador do utilizador ID, que é armazenado nos registos da base de dados e substituído a cada operação realizada. Novamente, no caso de uma mensagem retransmitida, esta verificação não será realizada e a mensagem retransmitida terá como resposta uma mesagem contento o valor NOK no campo $STATUS$. Novamente, ao receber tal mensagem, a aplicação web realizará a troca do servidor e solicitará um nova autenticação. A resistência a este ataque pode ser representada pelo caminho $C2.I \rightarrow C2.1 \rightarrow C2.2 \rightarrow C2.3 \rightarrow C2.8 \rightarrow C2.9 \rightarrow C2.10C2.10$ (figura 3.9). O mesmo raciocínio pode ser feito na retransmissão da mensagem $STATUS, ID, T+1, RS, E_{Ks}(T_{ID}), SIGNATURE_{Ks}$, para a aplicação web, com a diferença de que, neste caso, a verificação do campo $T+1$ é feita pela aplicação web.

Discussão Ressalta-se que uma combinação dos ataques descritos nesta seção podem ser utilizados por um servidor malicioso, mas é fácil perceber que os mecanismos utilizados para tolerá-los podem também ser combinados de forma a resistir tais ataques.

Como pode ser visto, o framework desenvolvido utilizou mecanismos para proteção da integridade (MAC e assinaturas digitais), cifras simétricas e números aleatórios gerados em cada operação e utilizados nas mensagens trocadas para implementar a segurança da aplicação web de forma a garantir o seu correto funcionamento, mesmo na presença de servidores web comprometidos. A funcionalidade de troca de servidor web tão logo seja constatado a presença de um servidor web ou um limite de tempo pré-definido seja atingido na espera das resposta permitiu tolerar os ataques possíveis de serem executados por um servidor web malicioso.

Capítulo 4

Implementação

Nesta seção, será descrito a implementação do TRIALs, no que se refere a sua organização geral, parâmetros de configuração, funcionalidades presentes na camada de segurança a serem utilizadas pelas aplicações web e camada de segurança a ser implementada na base de dados.

4.1 Organização Geral

O TRIALs foi desenvolvido na linguagem de programação JavaScript, com o uso da tecnologia AJAX e formato das mensagens em JSON. A estratégia foi criar uma camada de segurança adicional a esta tecnologia, permitindo a implementação de funcionalidades que garantissem a integridade e autenticidade dos dados recebidos de servidores web potencialmentes maliciosos, juntamente com a possibilidade de troca de servidor web tão logo tais propriedades de segurança sejam violadas ou um limite de tempo pré-definido seja atingido na espera de uma resposta a uma requisição HTTP. Esta camada adicional está divida em dois ficheiros JavaScript, a ser inluído na página principal da aplicação web: um relacionado com os parâmetros de configuração para o correto funcionamento do TRIALs e outro contendo as funcionalidades de segurança oferecidas pelo framework. Adicionalmente, uma camada de segurança a ser implementada na base de dados foi implementada para suportar a assinatura das mensagens enviadas por este componente para a aplicação web.

4.2 Configuração

Inicialmente, um conjunto de parâmetros devem ser configurados em um ficheiro JavaScript (*config.js*) para permitir o correto funcionamento do framework. A tabela 4.1 apresenta tais parâmetros, com suas respectivas descrições e valores iniciais.

Parâmetro	**Descrição**	**Valor Inicial**
ajaxRequest	Objeto XMLHttpRequest da API HTTP ECMAScript, responsável pela realização das requisições HTTP da aplicação web.	*new XMLHttpRequest()*
servers	Array contendo o endereço dos servidores web disponíveis. Os servidores serão utilizados na ordem que são inseridos na matriz.	Endereços dos servidores web a serem utilizados, inseridos pelo desenvolvedor da aplicação.
currentServer	Índice do servidor atualmente em uso pela aplicação web.	0
cert	Certificado digital, em formato X.509, para validação das assinaturas digitais feitas pela base de dados.	String contendo o certificado digital em formato X.509, que deve ser inserido pelo desenvolvedor da aplicação.
ID	Identificador do utilizador	*null*
P	*Hash* da palavra-passe fornecida pelo utilizador	*null*
Ks	Chave de sessão que será utilizada para cifrar dados nas comunicações entre a aplicação web e a base de dados. Seu valor será atualizado sempre que o protocolo de autenticação finalizar com sucesso.	*null*
T	Número aleatório, utilizado como *token* para realização das requisições da aplicação web. Seu valor será atualizado sempre que uma requisição for respondida com sucesso.	*null*
IV	Vetor inicial utilizado na cifra simétrica do algoritmo AES.	String com 16 bytes que deve ser inserido pelo desenvolvedor da aplicação.

Tabela 4.1: Parâmetros de configuração do TRIALs.

4.3 Camada de Segurança da Aplicação

Um outro ficheiro JavaScript (*seclayer.js*) foi utilizado para guardar as implementações das funções principais e secundárias do framework, que serão descritas a seguir . Define-se aqui como funções primárias aquelas que serão utilizadas diretamente pelos desenvolvedores da aplicação web e funções secundárias aquelas que serão usadas internamente pelo framework.

O framework utilizou duas bibliotecas JavaScript para a realização das operações criptográficas simétricas, validação de assinatura digital e função de síntese (*Hash*). São elas:

- *jsrsasign* [35]: biblioteca livre e de código aberto, desenvolvida por Kenji Urushima para realização e validação de assinaturas digitais utilizando o algoritmo RSA e os padrões de chave privada ASN.1, PKCS#1, PKCS#5 e de certificados digitais X.509. A biblioteca foi utilizada para a validação das assinaturas digitais das mensagens enviadas pela base de dados para a aplicação web, através de uma função secundária do framework que lê o certificado digital presente na variável global *cert*, recebe como parâmetros a mesagem e sua correspondente assinatura digital e retorna um valor booleano, de acordo com a validação ou não da assinatura digital.

- *SlowAES* [20]: biblioteca desenvolvida por Josh Davis e Alex Martelli que implementa o algoritmo AES para realização de cifras simétricas, disponível atualmente nas linguagens de programação JavaScript, Python, Ruby e PHP. A versão utilizada no framework foi a disponibilizada em JavaScript e permitiu a decodificação da variável global *T* presente nas mensagens de resposta da base de dados às requisições feitas pela aplicação. Este campo é necessário para o processamento das requisições da aplicação web e tolerância aos ataques de replay por servidores web maliciosos (seção 3.4). O framework implementou uma função secundária que recebe como parâmetros o númeto *T* cifrado, bem como a chave necessária para a realização da decodificação (parâmetro *Ks* do framework). A função utiliza então métodos da bilbioteca para a decodificação da cifra e retorna o corresponde texto em claro resultado da operação.

A funcionalidade de troca de servidor web foi implementada como uma função secundária que, a medida que é chamada, incrementa a variável global *currentServer*, permitindo o uso do próximo endereço de servidor web disponível na matriz *servers*. A função retorna um valor booleano, de acordo com a disponibilidade ou não de um servidor web na matriz e é chamada sempre que se constate a comunicação com um servidor web com falhas (maliciosas ou não).

O framework implementa ainda duas funções principais, que servem como uma API para utilização dos desenvolvedores de aplicações web que o desejem utilizar. Tais funções

correspondem à implementação dos protocolos de autenticação e de comunicação, descritos na seção 3.3.3 e ilustrados nas figuras 3.3 e 3.7, respectivamente. A tabela 4.2 apresenta as funções principais para os desenvolvedores das aplicações web, com suas respectivas descrições e parâmetros necessários.

Função	**Descrição**	**Parâmetros**
requestLogin	Função para realização da autenticação mútua entre o utilizador da aplicação e a base de dados	username, password, callback, fallback
requestData	Função para realizar as requisições de leitura e escrita para a base de dados, com garantia de integridade dos dados.	data, urlRequest, callback, fallback, transformJsonFunction

Tabela 4.2: Funções da API do TRIALs.

A função *requestLogin* implementa o protocolo de autenticação do TRIALs. Os parâmetros *username* e *password* representam, respectivamente, o identificador e a palavra-passe do utilizador que necessite se autenticar junto à base de dados. Estes dados podem, por exemplo, ser fornecidos pelo utilizador por meio de formulários HTML presentes na página principal da aplicação. Os parâmetros *callback* e *fallback* representam o nome de funções que devem ser implementadas pelo desenvolvedor da aplicação e que serão chamadas pelo framework para o caso da autenticação ser realizada com sucesso (função *callback*) ou no caso de ocorrer algum problema durante esta operação (função *fallback*). Um exemplo de uso desta função por uma aplicação web pode ser visto na listagem 4.1. Para simplificar, a inclusão de todos os ficheiros da biblioteca *jsrsasign* e *SlowAES* foram abreviados.

```
1  <html>
2  <head>
3   <!-- Ficheiros da biblioteca jsrsasign -->
4    <script src="javascript/rsa.js"></script>
5    <!--                .......                                  -->
6
7  <!-- Ficheiros da biblioteca SlowAES -->
8    <script src="javascript/aes.js"></script>
9    <!--                .......                                  -->
10
11 <!-- Ficheiros do FRAMEWORK -->
12   <script src="javascript/config.js"></script>
13   <script src="javascript/seclayer.js"></script>
14
15 <!-- Funcoes locais -->
16 <script>
17   function showHomePage() {
18     document.getElementById("loginDiv").style.display = "none";
```

```
19      document.getElementById("messageDiv").innerHTML = "";
20      document.getElementById("homePageDiv").style.display = "inline";
21    }
22
23    function badAuthentication() {
24      document.getElementById("messageDiv").innerHTML = "Bad
            Authentication. Try again.";
25    }
26
27    function doLogin() {
28      var username = document.getElementById("username").value;
29      var password = document.getElementById("password").value;
30      requestLogin(username, password, showHomePage, badAuthentication);
31    }
32  </script>
33  </head>
34  <body>
35  <div id="messageDiv" />
36
37  <div id="loginDiv">
38    <table width="30%" align="left" celpadding="1" cellspacing="1" border
        ="0">
39      <tr>
40        <td><b>Username: </b></td>
41        <td><input id="username" type="text"></td>
42      </tr>
43      <tr>
44        <td><b>Password: </b></td>
45        <td><input id="password" type="password"></td>
46      </tr>
47      <tr>
48        <td colspan="2" align="center"><input id="login_submit" type="
              button" value="Submit" onclick="doLogin()"></td>
49      </tr>
50    </table>
51  </div>
52
53  <div id="homePageDiv" style="display:none">
54  <!-- some content -->
55  </div>
56  </body>
57  </html>
```

Listagem 4.1: Exemplo de uso do protocolo de autenticação.

Como pode ser visto, a utilização do protocolo de autenticação pelo desenvolvedor de uma aplicação web resume-se em criar as funções de *callback* e *fallback*, bem como uma função que leia o identificador e a palavra-passe fornecida pelo utilizador e que realize a chamada da função *requestLogin* do framework com os parâmetros necessários (linhas 27 a 31 da listagem 4.1).

Como explicado na seção 3.3.3, finalizado o protocolo de autenticação, uma chave de sessão e um token para a próxima requisição são armazenados pela aplicação web e pela base de dados e o protocolo de comunicação (figura 3.7) pode ser iniciado. Este

protocolo foi implementado no framework pela função *requestData*. O parâmetro *data* diz respeito aos dados necessários para a realização de uma requisição. Pode ser, por exemplo, a chave primária de uma registo presente na base de dados a ser consultado, um objeto em formato JSON a ser inserido ou alterado pela base de dados ou até mesmo uma string vazia, quando, por exemplo, deseja-se somente listar todos os registos de uma tabela. O parâmetro *urlRequest* representa a complementação ao endereço do servidor web em utilização na qual se deseja direcionar a requisição, formando o que se conhece como recurso de um web service. Os parâmetros *callback* e *fallback* possuem a mesma funcionalidade do protocolo de autenticação.

Por fim, o parâmetro *transformJsonFunction* representa o nome da função JavaScript responsável pela transformação do objeto JSON que será recebido na resposta da requisição em uma string. Esta transformação é necessária pela facto do formato do resultado das consultas da base de dados ser diferente daquele recebido pela aplicação web, uma vez que o servidor web se encarregará de transformar todo o dado recebido pela base de dados para o formato JSON. Como os dados consultados pela base de dados fazem parte da mensagem resposta assinada digitalmente, faz-se necessária uma função que realize a transformação dos dados JSON recebidos para o formato utilizado pela base de dados de forma a garantir a correta validação da assinatura digital. Esta transformação pode ser implementada, por exemplo, realizando a simples concatenação dos valores dos atributos de um objeto armazenado. Por exemplo, supondo a utilização de uma base de dados relacional, contendo uma tabela que armazena registos dos utilizadores da aplicação, cujos atributos são *id*, *firstName*, *lastName* e *password*. Um registo desta tabela poderia conter os valores: *id* = ”alice”, *firstName* = ”Alice”, *lastName* = ”Stark”e *password* = ”41ic35t4rk”. Assim, a base de dados, ao responder a uma requisição para buscar os dados de tal registo, utilizaria na mensagem resposta a ser assinada a string ”aliceAliceStark41ic35t4rk”, juntamente com os demais campos da mensagem. Como o servidor web encaminhará para a aplicação web tal objeto utilizando o formato JSON, a função de transformação a ser implementada pelo desenvolvedor da aplicação web deve receber tal objeto JSON, extrair os valores dos seus campos e concatená-los, visando obter exatamente a mesma string utilizada pela base de dados para a realização da assinatura e viabilizar sua correta validação pelo TRIALs. Nos casos em que tal transformação não seja necessária, como por exemplo em requisições que não retornem objetos armazenados na base de dados (modificações ou inclusões de registos, por exemplo), este parâmetro deverá possuir valor igual a *null*.

A listagem 4.2 apresenta um exemplo do uso da função implementada no framework para a comunicação entre a aplicação web e a base de dados com garantia de integridade. Para simplificar, a exportação dos ficheiros das bibliotecas JavaScript utilizadas pelo framework, bem como dos ficheiros do próprio framework e as funções utilizadas no protocolo de autenticação foram omitidos do código.

```
1  <html>
2  <head>
3  <script>
4    function showUser(user) {
5      <!-- DOM manipulation to show user's attributes -->
6    }
7
8    function badRequest() {
9      document.getElementById("messageDiv").innerHTML = "Error Message";
10   }
11
12   function userTransform(user) {
13     var u = "";
14     u += user.uid +
15         user.firstName +
16         user.lastName;
17     return u;
18   }
19
20   function getUser(uid) {
21     var url = "/getUser?";
22     requestData(uid, url, showUser, badRequest, userTransform);
23   }
24 </script>
25 </head>
26 <body>
27 <!-- HTML Page showing users -->
28 </body>
29 </html>
```

Listagem 4.2: Exemplo de uso do protocolo de comunicação.

O exemplo apresenta a implementação de um funcionalidade de solicitação dos atributos de um determinado utilizador da aplicação. A linha 22 da listagem 4.1 realiza a chamada da função implementada pelo TRIALs para o início do protocolo de cominicação. Tal função, em caso de processamento correto da requisição, realizará uma chamada a função *userTransform* para permitir uma correta validação da assinatura digital feita pela base de dados e, ocorrendo tal validação, chamará a função *showUser*, passando como argumento o objeto JSON resultante da consulta feita pela base de dados e transformado pelo servidor web. Esta função realizará então as manipulações de objetos DOM necessárias para exibição do resultado da requisição para o utilizador. Caso a requisição não seja processada corretamente, a função *badRequest* será chamada para exibição de uma mensagem de erro.

Novamente, é possível observar que a utilização do framework para a realização de operações com garantia de integridade dos dados entre a aplicação web e a base de dados resume-se na leitura dos dados a serem utilizados na requisição, indicação do recurso a ser acessado, definição de uma função para tratamento dos dados recebidos, definição de uma função para o tratamento de erros e defnição de uma função de transformação do resultado da requisição, se for o caso.

O código fonte dos dois ficheiros JavaScript implementados do TRIALs estão listados no Anexo 1 do presente trabalho.

4.4 Camada de Segurança da Base de Dados

O TRIALs utiliza assinaturas digitais da base de dados como mecanismo de autenticidade e integridade das mensagens enviadas pela base de dados para a aplicação web. Assim, fez-se necessária uma camada de segurança adicional, impelementada na base de dados, de forma a garantir a realização dessas assinaturas de acordo com os protocolos de autenticação e comunicação implementados no framework.

Neste trabalho, uma camada de segurança foi adicionada a uma base de dados, com o objetivo de receber as requisições dos servidores web, processá-las realizando consultas/escritas dos dados da aplicação por meio do sistema de gestão de base de dados utilizado e realizar a assinatura digital das mensagens enviadas como respostas às requisições feitas.

Na aplicação web que foi desenvolvida utilizando o TRIALs para fins de avaliação neste trabalho, esta camada foi implementada utilizando a linguagem de programação Java, por meio de chamadas remotas de métodos de uma classe, utilizando a API Java RMI [19]. Tais métodos possuem como parâmetros a serem recebidos os valores dos campos presentes na mensagem JSON recebida pelo servidor web e realizam as validações e requisições à base de dados conforme fluxogramas ilustrados nas figuras 3.5 e 3.9. Processada a requisição, esta camada assina os dados a serem enviados como resposta para o servidor web, resposta esta que representa uma estrutura de dados (*ArrayList*) contendo os campos correspondetes às mensagens de resposta definidos nos protocolos implementados, a saber: $STATUS$, ID, $E_{K_s}(T)$, S_a, S_b e $SIGNATURE_{K_s}$ para o caso do protocolo de autenticação; e $STATUS$, ID, $T+1$, RS, $E_{K_s}(T_{ID})$ e $SIGNATURE_{K_s}$ para o caso do protocolo de comunicação. Ressalta-se que, como mencionado na seção 3.3, para o protocolo de comunicação, pode-se utilizar em substuição da assinatura digital um MAC da mensagem resposta utilizando a chave de sessão estabelecida no protocolo de autenticação.

Capítulo 5

Validação e Avaliação

Esta seção apresentará as avaliações de desempenho realizadas em uma aplicação desenvolvida a partir do TRIALs, no que se refere aos tempos de processamento de algumas das funcionalidades implementadas, seja pela base de dados, seja pela aplicação web em execução no navegador web de um utilizador.

Inicialmente, será apresentada a validação do TRIALs por meio de uma aplicação web exemplo desenvolvida a partir do framework. A seguir serão apresentados os resultados de avaliações realizadas, onde testes de desempenho foram realizados na base de dados e na aplicação web de forma isolada, levando em consideração as operações de validação de assinaturas digitais e MAC em navegadores web e o tempo de processamento local dos protocolos de autenticação e de comunicação pela base de dados, utilizando diferentes tamanhos de mensagens respostas e de mecanismos de integridade (assinatura digital e MAC). Em seguida, serão apresentados os resultados das avaliações que referem-se às comparações feitas entre as latências dos protocolos de autenticação e comunicação entre uma aplicação desenvolvida a partir do TRIALs e a aplicação exemplo, sem a presença de falhas em servidores web. Como consequência desta última, uma estimativa do *throughput* de um servidor de base de dados será apresentada, bem como do *overhead* do uso do TRIALs em um ambiente de rede de larga escala. Por fim, serão apresentados os resultados observados nas latências dos protocolos de autenticação e comunicação, para o caso de presença de falhas nos servidores web. Todos os testes apresentados foram realizados utilizando 10.000 amostras, dos quais representamos a média aritmética.

5.1 Validação: Aplicação Web Exemplo

Com a finalidade de validar a possibilidade do desenvolvimeto de uma aplicação web utilizando o framework proposto neste trabalho, foi desenvolvida uma pequena aplicação web que representa um sistema informático de uma loja de venda de miniaturas de carros e aviões, cujo modelo de dados pode ser visto na figura 5.1.

A este modelo de dados foi acrescentado uma tabela para armazenamento dos dados

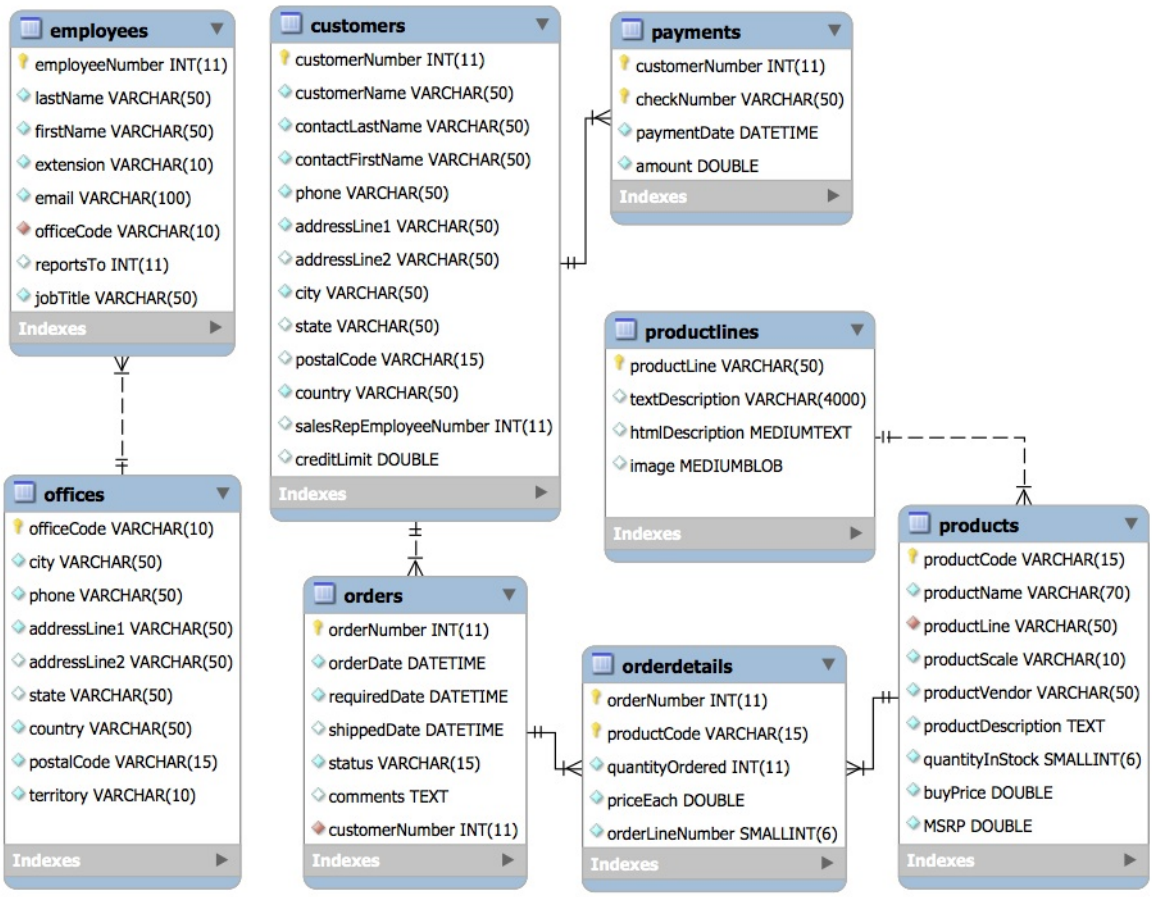


Figura 5.1: Modelo de dados da aplicação web exemplo.

dos utilizadores da aplicação, necessários para o funcionamento do framework (ID, P, K_s e T). Foram desenvolvidos também dois servidores web, um utilizando a linguagem de progamação Java e outro utilizando o PHP, o que permitiu a realização de testes da funcionalidade de troca de servidores em caso de falhas. A escolha de linguagens de programação diferentes visou demonstrar a independência do framework com relação a arquitetura utilizada pelos servidores web disponíveis, além de usufruir das vantagens presentes no mecanismo da diversidade para reduzir a probabilidade de vulnerabilidades existentes em mais de uma componente de um sistema tolerante à falhas [6]. Este mecanismo é utilizado de forma que cada componente use diferentes software para executar a mesma função, com a expectativa de que as diferenças irão reduzir a ocorrência de vulnerabilidades comuns, isto é, as vulnerabilidades existentes em mais de um sistema [17].

A parte do servidor de aplicação web foi desenvolvida utilizando a versão JDK 1.6 e foi concretizado usando o servidor HTTP Apache Tomcat 6, Java Beans e Servlets, enquanto o servidor PHP utilizou a versão 5.5 e foi concretizado usando o servidor HTTP

Apache 2.4. Para a base de dados, foi utilizado o Sistema de Gestão de Base de Dados MySQL versão 5.6, onde foram criadas as tabelas do modelo de dados da figura 5.1, com a inserção de alguns registos.

5.2 Avaliação do TRIALs

5.2.1 Primitivas Criptográficas no Navegador Web

Com o objetivo de avaliar o tempo de execução da validação de assinaturas digitais em navegadores web, foram realizados testes em diferentes navegadores web executando tal operação. Para isso, uma pequena página web estática foi construída, contendo um formulário com a mensagem e sua respectiva assinatura, bem como o necessário certificado digital pré-configurado como uma variável global em um script JavaScript. O algoritmo para verificação das assinaturas digitais utilizado foi o RSA, com chaves de 512 e 1024 bits. A implementação do RSA em JavaScript foi realizada através da biblioteca *jsrsasign* [35], a mesma utilizada no framework TRIALs. As médias dos testes realizados, com diferentes tamanhos de mensagens, estão apresentadas na figura 5.2. As figuras 5.2(a) e 5.2(c) representam as médias observadas em mensagens de até 1 MB e as figuras 5.2(b) e 5.2(d) apresentam os mesmos dados, porém com o limite superior do intervalo do tamanho das mensagens igual a 100 KB, com o objetivo de observar melhor o desempenho dos navegadores neste intevalo.

A avaliação dos gráficos das figuras 5.2(b) e 5.2(d) permite verificar que o tempo de processamento na validação de assinaturas digitais são próximos para tamanhos de mensagem inferiores a 100 KB, não ultrapassando os 120 ms, tanto para o uso de chaves de 512 bits, quanto de 1024 bits. A partir deste valor, verifica-se uma grande variação no desempenho dos navegadores, sendo o Mozilla Firefox o que apresenta melhor desempenho. Observa-se também que, com exceção do navegador Internet Explorer, todos os tempos de processamento na validação das assinaturas digitais de mensagens inferiores a 1 MB são inferiores a 1.000 ms (figuras 5.2(a) e 5.2(c)), o que pode ser considerado como um tempo aceitável para a realização desta funcionalidade em navegadores web. Observa-se também um pequena diferença de desempenho na realização de assinaturas com chaves de 512 bits ou 1024 bits. Esta análise encoraja o desenvolvimento de aplicações web que utilizem a validação de assinaturas digitais pelos navegadores web dos seus utilizadores, o que permitiria a identificação de servidores web ou atacantes que alteram as mensagens trocadas entre uma aplicação web e uma base de dados ou até mesmo outras aplicações web.

Outra avaliação realizada com navegadores web foi a decifra de conteúdos utilizando cifras simétricas. O protocolo utilizado para avaliação foi o mesmo utilizado pelo TRIALs: o AES. A implementação do AES utilizou a biblioteca SlowAES [20], com chaves de 256 bits e modo de operação CBC (*Cipher Block Chaining*) [23]. A exemplo

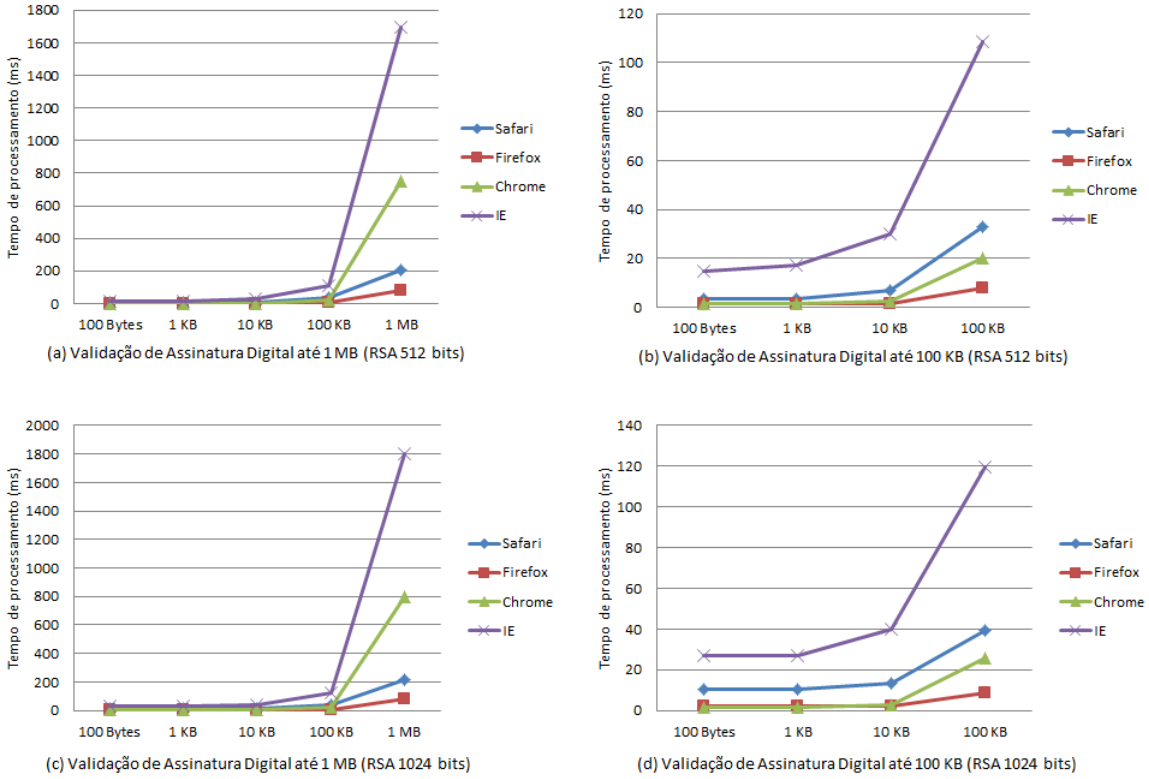


Figura 5.2: Tempo de validação de assinaturas digitais RSA.

da avaliação anterior, também foi construída uma pequena página web estática para a realização dos testes de desempenho, utilizando diferentes tamanhos de mensagens. As médias dos testes estão apresentadas na figura 5.3. Ressalta-se que o TRIALs utiliza cifra simétrica apenas para proteger o *token* utilizado nas mensagens trocadas na interação com a base de dados, gerados como números aleatórios, logo, em mensagens pequenas.

É possível observar no gráfico da figura 5.3 que o tempo para a realização da decifra de mensagens são semelhantes nos navegadores web testados, não ultrapassando este tempo os 130 ms. Estes resultados também apresentam-se como favoráveis à utilização de navegadores web para garantir a confidencialidade de mensagens pequenas de aplicações web utilizando cifras simétricas.

O TRIALs utiliza também MACs como mecanismo de verificação de integridade de mensagens, sendo o algoritmo *hash* o SHA-256 [1] o escolhido na concretização. Por isso, foram realizados também testes de desempenho de navegadores web para a geração de MAC utilizando tal protocolo. Novamente, os testes foram realizados a partir de uma pequena página web estática criada, onde mensagens de diferentes tamanhos foram utilizadas juntamente com uma chave de 256 bits para geração dos MACs. As médias dos testes realizados são mostradas na figura 5.4. A figura permite verificar que o tempo

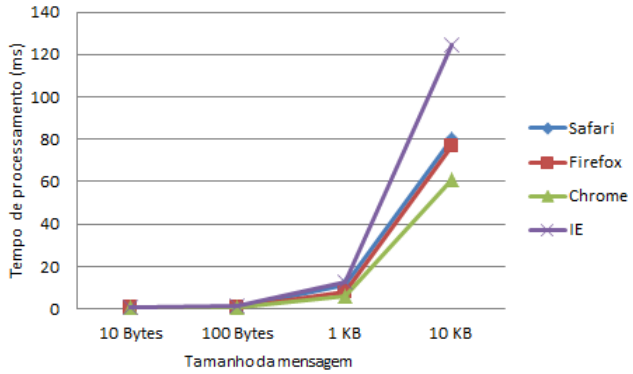


Figura 5.3: Tempo de decifra com cifra simétrica AES-256.

de processamento para geração de MAC em mensagens de até 100 KB são semelhantes entre os navegadores web testados e que não ultrapassam 200 ms. A partir deste limite, é possível verificar diferentes desempenhos entre os navegadores, destacando-se como o mais eficiente o Mozilla Firefox. Novamente, esta análise encoraja a utilização de navegadores web para verificar a integridade de mensagens em aplicações web utilizando o mecanismo de MAC, uma vez que os tempos de processamento apresentados, principalmente para mensagens inferiores a 100 KB, mostram-se viáveis a sua utilização sem prejudicar o desempenho da aplicação web, o que permite que aplicações web em execução nos navegadores web possam identificar a alteração de mensagens por atacantes ou servidores web maliciosos.

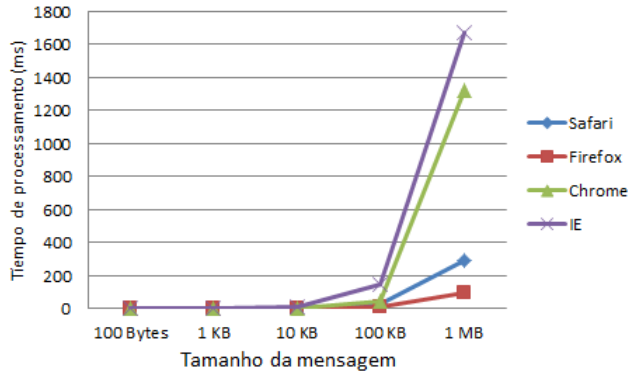


Figura 5.4: Tempo de geração de *hash* com SHA-256.

5.2.2 Primitivas Criptográficas no Servidor da Base de Dados

Do lado da base de dados, foram realizadas avaliações no tempo de processamento local dos protocolos de autenticação e de comunicação implementados no TRIALs, sem levar em consideração a latência de uma rede de dados (que será avaliada na seção 5.2.3). Os testes foram realizados utilizando assinatura digital por meio do RSA (chaves de 512 e 1024 bits) e MAC utilizando o SHA-256 (chave de 256 bits). Além disso, a aplicação web *baseline*, sem os mecanismos de segurança implementados pelo TRIALs, também foi avaliada no que se refere ao seu desempenho, com o objetivo de realizar uma comparação com a aplicação exemplo desenvolvida e identificar o *overhead* dos mecanismos propostos.

A base de dados utilizada nos testes foi instalada em uma máquina contendo 1 CPU Intel Core i7, com 4 *cores*, de 1,73 GHz cada e memória RAM de 8 GB. A camada de segurança da base de dados descrita na seção 4.4 foi implementada na linguagem de programação Java, com o objetivo de realizar consultas a uma base de dados instalada no sistema de gestão de base de dados MySQL 5.5 por meio da API JDBC [15].

Para a realização dos testes de desempenho do protocolo de autenticação, uma classe Java implementou a sequência de atividades ilustradas no fluxograma da figura 3.5. Mais precisamente, foi medido o tempo de processamento do caminho $A2.I \rightarrow A2.1 \rightarrow A2.2 \rightarrow A2.3 \rightarrow A2.4 \rightarrow A2.5 \rightarrow A2.6 \rightarrow A2.7 \rightarrow A2.9$ da citada figura. Além disso, foram realizados testes utilizando a aplicação *baseline*, no qual a base de dados realiza apenas a verificação do identificador do utilizador e sua palavra-passe. O resultado dos testes do protocolo de autenticação estão representados na tabela 5.1.

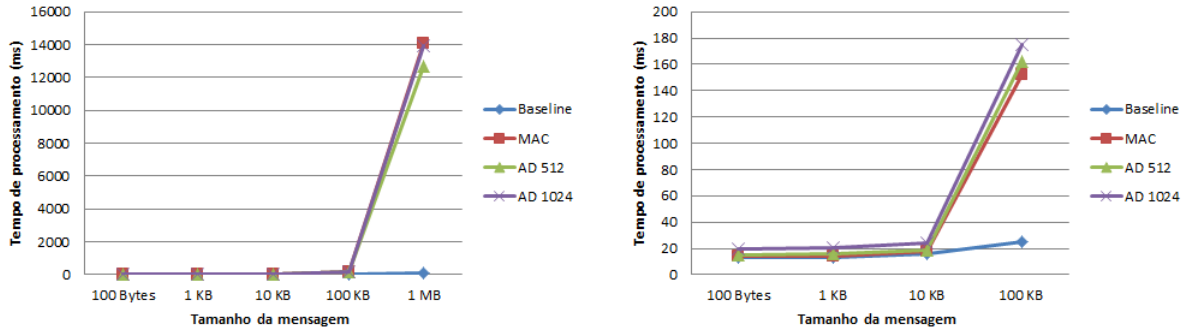
	Baseline	**TRIALs Servidor (512 bits)**	**TRIALs Servidor (1024 bits)**
Média	14,05 ms	16,75 ms	21,9 ms
Desvio Padrão	3,00 ms	3,14 ms	3,48 ms
Mediana	13 ms	16 ms	22 ms
95 Percentil	16 ms	18 ms	23 ms

Tabela 5.1: Tempo de processamento do protocolo de autenticação.

Pode-se verificar pelos resultados dos testes de desempenho do protocolo de autenticação que o aumento no tempo de processamento decorrente da implementação das funcionalidades de segurança do TRAILs pode ser considerado inócuo, se comparados com os benefícios trazidos pelo mesmo, a saber, a capacidade de uma aplicação web interagir de forma segura com uma base de dados, mesmo na presença de servidores web não confiáveis.

Foram realizados também testes locais na base de dados para verificação do tempo de processamento do protocolo de comunicação implementado pelo TRIALs. De forma similiar ao teste anterior, testes de desempenho também foram realizados com uma aplica-

ção *baseline*, a fim de realizar a comparação desta com a aplicação exemplo. Os testes foram realizados variando o tamanho das mensagens de resposta da base de dados, bem como o mecanismo de integridade utilizado: assinaturas digitais (RSA com chaves de 512 e 1024 bits) e MAC (SHA-246 com chave de 256 bits). O teste pretendeu simular um pedido da aplicação para realização de uma consulta a todos os registos de uma tabela armazenada na base de dados. Os tamanhos das mensagens foram variando de acordo com o número de registos presentes na tabela em questão. A figura 5.5 apresenta os resultados obtidos nos testes. Os dados utilizados em ambos os gráficos são os mesmos, estando a figura 5.5(b) apenas limitando o intervalo do tamanho das mensagens para até 100 KB, com o objetivo de melhor observar o desempenho da base de dados neste intervalo.



(a) Tempo de processamento até 1 MB

(b) Tempo de processamento até 100 KB

Figura 5.5: Tempo de processamento do protocolo de comunicação.

A análise do gráfico da figura 5.5(a) permite verificar que, para mensagens maiores que 100 KB, aplicações desenvolvidas com o framework TRIALs aumentam significamente o tempo de processamento do protocolo de comunicação pela base de dados, chegando a mais de 12 segundos para mensagens de 1 MB, qualquer que seja o mecanismo de integridade utilizado. Esta observação nos leva a crer que aplicações web projetadas para lidar com mensagens maiores que 100 KB não são candidatas a serem desenvolvidas utilizando o TRIALs, uma vez que o tempo das respostas às requisições a inviabilizam. Entretanto, ao restringirmos o limite máximo do tamanho da mensagem para 100 KB, como pode ser visto no gráfico da figura 5.5(b), observa-se que o uso de mensagens até 10 KB no protocolo de comunicação apresenta tempos de processamento muito próximo daquele associado à aplicação *baseline*, sendo seu valor não superior a 50 ms. Para mensagens entre 10 KB e 100 KB, é possível verificar uma queda de desempenho da aplicação exemplo com relação à aplicação *baseline*, independente do mecanismo de integridade utilizado. Entretanto, esta queda de desempenho não ultrapassa o tempo de processamento de 180 ms, o que pode ser considerado um tempo de processamento aceitável para aplicações web com as funcionalidades de segurança fornecidas pelo TRIALs.

Em seguida, com o objetivo de estimar o *throughput* do processamento das mensagens de uma aplicação web desenvolvida a partir do TRIALs comparado com a aplicação *baseline*, foi utilizada a média dos tempos de processamento das mensagens de 100 bytes, 1 KB e 10 KB utilizadas na figura 5.5, considerando que a maioria das aplicações web utilizam mensagens com tamanhos neste intervalo. Tal média foi utilizada para calcular o número de mensagens processadas em 1 segundo, por cada *core* da CPU do computador utilizado como servidor da base de dados. Este valor foi então utilizado como base para estimar o *throughput* de máquinas com diferentes número de *cores*, conforme apresentado na figura 5.6.

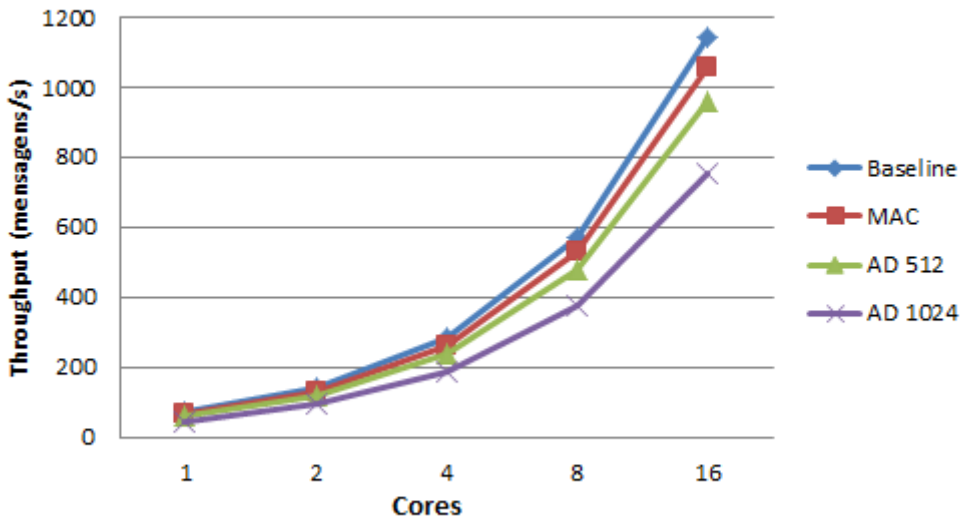


Figura 5.6: Throughput do protocolo de comunicação.

A análise da figura 5.6 permite observar que os *throughputs* do processamento das mensagens com uso do mecanismo de MAC e assinaturas digitais com chaves de 512 bits têm valores próximos ao processamento das mensagens sem este mecanismo, presente na aplicação *baseline*. Tal observação corrobora para o facto de que o *overhead* de processamento introduzido pela TRIALs pode ser considerado pequeno, se comparado com os benefícios de segurança que o mesmo introduz nas aplicações web desenvolvidas a partir do framework.

5.2.3 Avaliação na Ausência de Falhas

Nesta seção serão apresentados os resultados obtidos nos testes para analisar a latência total dos protocolos de autenticação e de comunicação implementados no TRIALs na

ausência de falhas dos servidores web. Novamente, a aplicação *baseline* foi utilizada como forma de comparação e verificação do *overhead* acrescentado pelas funcionalidades de segurança oferecidas pelo framework.

Para esta avaliação foram utilizadas três máquinas distintas, distribuídas em uma LAN (*Local Area Network*): um cliente, responsável pelas requisições ao servidor web; um servidor web, implementado utilizando o software Apache Tomcat 6 e servlets; e uma base de dados, implementado conforme descrito na seção 5.2. A arquitetura da avaliação realizada está representada na figura 5.7. Enquanto a comunicação entre a máquina cliente e o servidor web ocorre por meio do protocolo HTTP, a comunicação entre o servidor web e a base de dados foi implementada utilizando Java RMI [19]. Cabe ressaltar que a latência na comunicação entre as máquinas na rede local utilizada foi de 0,1 ms.

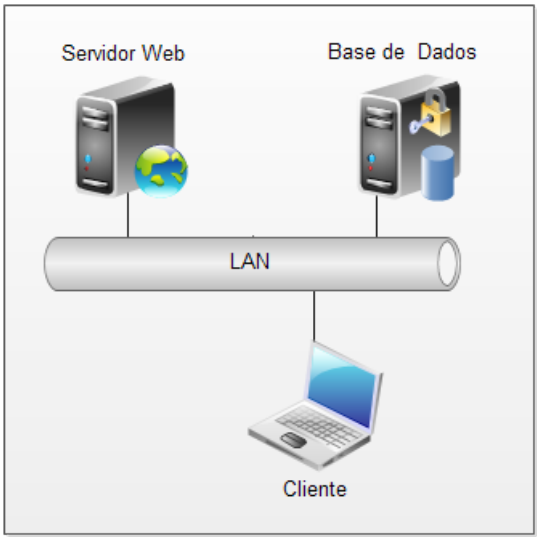


Figura 5.7: Arquitetura da avaliação.

Para avaliação, foi utilizada a ferramenta httperf [26], que permite avaliar o desempenho de servidores web a partir de comandos que geram diversas requisições HTTP, agrupando os resultados e exibindo-os em forma de dados estatísticos. Assim, a avaliação do protocolo de autenticação foi realizada enviando 10.000 mil requisições de autenticação a partir da máquina cliente para o servidor web, que por sua vez, comunicava-se com a base de dados através da chamada de um método remoto, que realiza a sequência das atividades ilustradas no fluxograma da figura 3.5. As médias das latências observadas nos testes estão apresentadas na tabela 5.2.

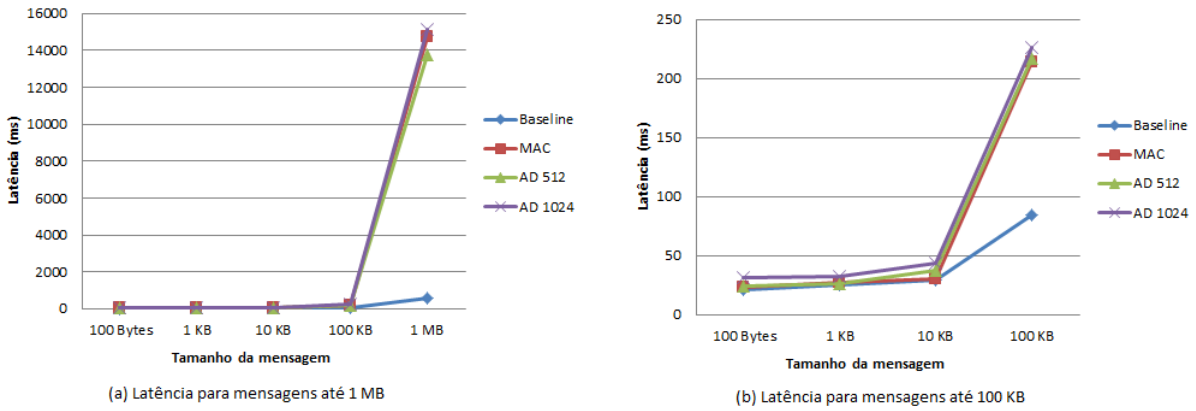
De forma semelhante ao observado na avaliação da seção 5.1, é possível verificar que o *overhead* adicionado pela utilização do TRIALs pode ser considerado pequeno, uma vez que tais valores não são perpecptíveis pelos utilizadores da aplicação web. Esta observação permite o desenvolvimento de aplicações web com um mecanismo de autenticação mais robusto do que o tradicional utilizador/palavra-passe, sem que a

	Baseline	TRIALs (512 bits)	TRIALs (1024 bits)
Média	16,9 ms	29,0 ms	35,4 ms
Desvio padrão	2,02 ms	3,52 ms	4,9 ms
Mediana	16 ms	14 ms	29 ms
95 Percentil	18 ms	32 ms	39 ms

Tabela 5.2: Latência do protocolo de autenticação.

experiência do usuário seja significamente alterada.

Para avaliação do protocolo de comunicação, a exemplo de como foi realizada a avaliação local da base de dados, foram geradas diversas requisições HTTP para a aplicação web exemplo que representava uma consulta a todos os registos de uma determinada tabela da base de dados. O tamanho das mensagens de respostas foram sendo variadas aumentando o número de registos da tabela em questão, bem como os mecanismos de integridade da mensagem resposta (assinatura digital e MAC). As médias obtidas na avaliação estão representadas na figura 5.8.



(a) Latência para mensagens até 1 MB

(b) Latência para mensagens até 100 KB

Figura 5.8: Latência do protocolo de comunicação.

De forma semelhante à avaliação realizada localmente na base de dados, é possível verificar no gráfico da figura 5.8(a) que, para mensagens de 1 MB, a latência do protocolo de comunicação é superior a 12 segundos, o que pode ser considerado inviável para aplicações web. Entretanto, a análise do gráfico 5.8(b) permite observar que, para mensagens de até 10 KB, a latência apresenta valores bem próximos daquela medida para a aplicação *baseline*, com valores inferiores a 50 ms. Isto permitiria o desenvolvimento de aplicações web com as funcionalidades de segurança oferecidas pelo TRIALs, cuja experiência do utilizador fique bem próxima a de uma aplicação sem tais funcionalidades. Para mensagens entre 10 KB e 100 KB, o desempenho da aplicação desenvolvida a partir do TRIALs apresenta menor desempenho, porém, sem ultrapassar a latência de 250

ms, o que pode ser considerado um *overhead* aceitável para aplicações web que garantem uma interação segura com uma base de dados, mesmo na presença de servidores web maliciosos.

Visando estimar o *overhead* da utilização do TRIALs em uma ambiente de rede de larga escala usando como referência a aplicação web *baseline*, bem como compará-lo com o *overhead* observado na rede local onde foram realizados os testes de latência deste trabalho, foram utilizados os dados médios de latência de rede WAN apresentados em [39]. Estes valores representam a latência média na comunicação entre máquinas distribuídas em uma WAN na Europa (40 ms) e na América do Norte (70 ms). A tabela 5.3 apresenta os resultados estimados.

	Local	**Europa (40 ms)**	**América do Norte (70 ms)**
MAC	6%	2%	2%
Assinatura Digital 512 bits	19%	7%	5%
Assinatura Digital 1024 bits	43%	17%	11%

Tabela 5.3: *Overhead* do TRIALs.

Os dados apresentados na tabela 5.3 mostram que, em ambientes com latência de comunicação maiores, o *overhead* acrescentado pela utilização do TRIALs no desenvolvimento de aplicações web não ultrapassa os 20 %, o que pode ser considerado como aceitável.

5.2.4 Avaliação na Presença de Falhas

Com a finalidade de observar o desempenho de uma aplicação desenvolvida a partir do TRIALs na presença de falhas em servidores web, foram realizados testes que simularam a existência de um servidor web malicioso, que realizava alterações dos dados recebidos pela base de dados, antes de seu envio para o navegador web após uma requisição. Esta alteração, tanto no protocolo de autenticação quanto no protocolo de comunicação, é identificada pela aplicação web em execução no navegador web do utilizador pela validação da assinatura digital feita pela base de dados. Não ocorrendo tal validação, a funcionalidade do TRIALs de troca de servidores é invocada e uma nova requisição é realizada para o próximo servidor web disponível, até que um servidor correto seja contatado.

Para o teste, foram utilizados dois servidores web: um concretizado com o servidor de aplicação Apache Tomcat 6, utilizando a linguagem de programação Java; e outro com o servidor de aplicação Apache 2, utilizando a linguagem de programação PHP, porém com recursos computacionais semelhantes. Ambos os servidores web contatavam uma mesma base de dados e a troca de servidores foi realizada de forma transparente para o utilizador.

Neste teste (dois servidores web disponíveis), a expectativa na latência dos protocolos de autenticação e de comunicação em operações na presença de servidores web malicio-

sos era de que a mesma atingisse o valor próximo ao dobro do observado em operações sem a ocorrência de falhas, uma vez que, a menos da função de troca de servidor, implementada com um simples incremento de uma variável global, nenhuma funcionalidade adicional é realizada nesta operação. De facto, apesar da diferença entre os ambientes dos servidores web, os testes realizados apresentaram médias muito próximas ao dobro dos valores apresentados na tabela 5.2 e na figura 5.8.

5.3 Considerações finais

Exposta neste capítulo a validação e avaliação do TRIALs, é possivel verificar que o desenvolvimento de aplicações web utilizando o framework é exequível e o desempenho das mesmas, levando em consideração as funcionalidades de segurança acrescentadas pelo framework ,mostrou-se como viável para o caso do tamanho das mensagens trocadas entre a aplicação web e a base de dados estarem limitadas até 100 KB.

Foi possível obervar também a viabilidade da utilização de navegadores web para a realização de validações de assinaturas digitais com chaves de 512 ou 1024 bits, bem como a decifra de segredos utilizando cifra simétrica e a garantia de integridade com o mecanismo de MAC, sem que seja necessária a instalação de software adicional nos *desktops* dos utilizadores e sem que isso afete de forma significativa a experiência do utilizador da aplicação no que se refere ao seu desempenho. Adicionalmente, permitiu-se prever que a utilização de aplicações web desenvolvidas a partir do TRIALs em ambientes de rede de larga escala acarretará em um *overhead* insignificante, permitindo sua utilização em aplicações web reais.

Cabe ressaltar que o TRIALs foi desenvolvido para fortalecer o conceito de aplicações web com fortes características de aplicações *desktops*, com extensivo uso dos navegadores web para execução de funções através da linguagem de programação JavaScript, interagindo com servidores web concretizados como web services.

Capítulo 6

Conclusão

Este relatório apresenta o trabalho realizado no desenvolvimento do TRIALs (*Trusted Rich Internet Application Layers*), um framework para programação de aplicações web seguras, permitindo uma interação segura com uma base de dados, mesmo a partir de interações com servidores web não confiáveis. O TRIALs foi desenvolvido utilizando a tecnologia AJAX e mecanismos de segurança que permitam a garantia das propriedades de autenticidade e integridade dos dados de uma aplicação web. Além disso, foi implementada uma funcionalidade de troca de servidor web tão logo tais propriedades de segurança sejam violadas ou um tempo limite na resposta de um servidor web seja atingido.

Inicialmente, este trabalho foi motivado pela proteção de aplicações web contra ataques de desfiguração de páginas eletrônicas, onde um atacante altera os dados presentes na mesma com o objetivo de indisponibilizar os seus serviços, redirecionar o utilizador para outras páginas contendo scripts maliciosos ou até mesmo prejudicar a imagem da organização proprietária da página eletrônica. Durante o desenvolvimento do trabalho, identificou-se que a utiliziação de aplicações web com fortes características de aplicações *desktops*, interagindo com web services para obtenção de dados, é, na verdade, uma alternativa à tradicional solução de utilização de servidores web executando CGI. Esta alternativa faz com que o navegador web assuma um importante papel no processamento dos dados recebidos de um servidor web, não limitando-se a apenas renderizá-los para sua exibição, mas também a tratá-los de forma a realizar validações e exibí-los de acordo com as particularidades da aplicação web em questão. Cabe ressaltar que apesar do TRIALs ter sido desenvolvido com foco no desenvolvimento de aplicações web, o seu modelo pode ser extendido para outros tipos de aplicações, como, por exemplo, para dispositvos móveis e voltadas para *desktops*.

A arquitetura do TRIALs utiliza a premissa da utilização de uma base de dados confiável, que garante que os dados fornecidos para a aplicação web estão corretos e que, caso sejam modificados na transmissão, tais modificações sejam possíveis de serem identificadas pela aplicação. Tal garantia foi concretizda com os mecanismos de assina-

tura digital e MAC, que podem ser validadas por navegadores web sem a necessidade de instalação de software adicional ou mudanças nos servidores web. Para tal, foram implementados dois protocolos no framework desenvolvido, que funcionam como API para desenvolvedores de aplicações web: um de autenticação mútua e outro de comunicação com garantia de integridade. Tais protocolos foram implementados em forma de funções JavaScript, facilmente utilizáveis no desenvolvimento de aplicações web.

O TRIALs oferece uma camada adicional de segurança à aplicações web que as tornem tolerantes a ataques de servidores web comprometidos como: alterações nos códigos da aplicação, atraso nas respostas às requisições, alterações nas mensagens e retransmissão de mensagens.

A avaliação de desempenho do TRIALs identificou a possibilidade da utilização de um mecanismo de autenticação mais robusto do que o tradicional utilizador/palavra-passe, com um *overhead* pequeno e imperceptível para o utilizador, atigindo uma latência média nos testes de, no máximo, 35 ms em rede local. Para a garantia de integridade na comunicação entre a aplicação web e a base de dados, o protocolo implementado mostrou-se eficiente para mensagens de até 100 KB, onde as latências médias não ultrapassaram os 250 ms em uma rede local e um valor percentual estimado em uma rede de larga escala de, no máximo, 20% em relação à aplicação *baseline*.

O trabalhou trouxe como importante contribuição a possibilidade de uso de navegadores web para executar funções de síntese e de criptografias simétricas e assimétricas, como forma de aumentar a segurança de aplicações web que necessitem interagir com servidores web não confiáveis. Foram realizadas avaliações da execução dessas funções em diversos navegadores web, onde foi possível verificar a viabilidade do seu uso em mensagens de até 100 KB.

A avaliação mostrou também a inviabilidade do uso do TRIALs para aplicações web com necessidade de troca de mensagens com a base de dados com tamanho superiores a 100 KB, onde foi possível observar uma latência média de até 14 segundos para mensagens de 1 MB em rede local.

Como trabalhos futuros, sugere-se a avaliação do TRIALs em uma ambiente de rede de larga escala, de forma a confirmar as estimativas feitas neste ambiente. Sugere-se também pesquisas que visem adaptar as já conhecidas soluções de serviços confiáveis, mesmo na presença de réplicas bizantinas, aos protocolos implementados pelo TRIALs. Por fim, sugere-se a concretização da camada de segurança da base de dados proposta pelo TRIALs para um ambiente de armazenamento utilizando uma nuvem de nuvens e que seja tolerante à faltas bizantinas, o que está sendo desenvolvido no âmbito de uma tese de doutaramento no DI-FCUL.

Apêndice A

Código fonte do TRIALs

```
1  //The XML HTTP Request Object
2  var ajaxRequest = new XMLHttpRequest();
3
4  //The Web Servers's address
5  var servers = new Array();
6  servers[0] = "http://10.10.5.94:8080/jwebservice";
7  servers[1] = "http://10.10.5.94/webservice";
8
9  //servers[0] = "http://localhost/webservice";
10 //servers[1] = "http://localhost:8080/jwebservice";
11
12 var currentServer = 0;
13
14 //The Database's x509 Certificate (512 bits)
15 var cert = "-----BEGIN CERTIFICATE-----\
16 MIIDCzCCArWgAwIBAgIJAMrJd1jTb1cKMA0GCSqGSIb3DQEBBQUAMIGNMQswCQYD\
17 VQQGEwJQVDEPMA0GA1UECBMGTGlzYm9uMQ8wDQYDVQQHEwZMaXNib24xCzAJBgNV\
18 BAoTAlVMMQ0wCwYDVQQLEwRGQ1VMMRowGAYDVQQDExFBbmRlcnNvbiBCYXJyZXR0\
19 bzEkMCIGCSqGSIb3DQEJARYVYWJhcnJldHRvNzhAZ21haWwuY29tMB4XDTEzMDMy\
20 NTE4MzQzNVoXDTEzMDQyNDE4MzQzNVowgY0xCzAJBgNVBAYTAlBUMQ8wDQYDVQQI\
21 EwZMaXNib24xDzANBgNVBAcTBkxpc2JvbjELMAkGA1UEChMCVUwxDTALBgNVBAsT\
22 BEZDVUwxGjAYBgNVBAMTEUFuZGVyc29uIEJhcnJldHRvMSQwIgYJKoZIhvcNAQkB\
23 FhVhYmFycmV0dG83OEBnbWFpbC5jb20wXDANBgkqhkiG9w0BAQEFAANLADBIAkEA\
24 y5Nxl5c3t4nXiCXGayTochTMdT/pTp1kbjV6qC1L/kXweSjGlypg9u6I2SdenG15\
25 UTETxjB2NERSeh4l+IKcCQIDAQABo4H1MIHyMB0GA1UdDgQWBBQ57mjCWWUyCV0+\
26 wg8Qfl8GWmhxzDCBwgYDVR0jBIG6MIG3gBQ57mjCWWUyCV0+wg8Qfl8GWmhxzKGB\
27 k6SBkDCBjTELMAkGA1UEBhMCUFQxDzANBgNVBAgTBkxpc2JvbjEPMA0GA1UEBxMG\
28 TGlzYm9uMQswCQYDVQQKEwJVTDENMAsGA1UECxMERkNVTDEaMBgGA1UEAxMRQW5k\
29 ZXJzb24gQmFycmV0dG8xJDAiBgkqhkiG9w0BCQEWFWFiYXJyZXR0bzc4QGdtYWls\
30 LmNvbYIJAMrJd1jTb1cKMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQEFBQADQQAu\
31 uRqdpLOuR8j1ykCXc1TB1M/oGNCG2a0K35/VbDe8LuxRXQu5UANDs1n+6mLGieNa\
32 76vgy7cc2kHtWYdmySAp\
33 -----END CERTIFICATE-----";
34
35 //Initial server
36 var baseUrl = servers[0];
37
38 // The Client's ID
39 var ID;
40
```

```
41 // The Password's Hash
42 var P;
43
44 //Session Key;
45 var Ks;
46
47 //Token
48 var T;
49
50 //Initial Vector to AES Decode
51 var iv = "1234567890123456";
52
53 //Change server
54 function changeServer() {
55   if((currentServer+1) != servers.length) {
56     currentServer++;
57     baseUrl = servers[currentServer];
58     return true;
59   }
60   else {
61     return false;
62   }
63 }
64
65 //RSA Sign Validation
66 function doVerify(sMsg, hSig) {
67     var x509 = new X509();
68     x509.readCertPEM(cert);
69     var isValid = x509.subjectPublicKeyRSA.verifyString(sMsg, hSig);
70     return isValid;
71 }
72
73 //AES Decode
74 function aesDecode(cypher, key) {
75
76   var ivByteArray = cryptoHelpers.convertStringToByteArray(iv);
77   var keyByteArray = cryptoHelpers.convertStringToByteArray(key);
78   var bytesToDecrypt = cryptoHelpers.base64.decode(hexToBase64(cypher))
       ;
79   //document.write(bytesToDecrypt + "<br>");
80
81   var decrypt = slowAES.decrypt(bytesToDecrypt,
82   slowAES.modeOfOperation.CBC,
83   keyByteArray,
84   ivByteArray);
85
86   var plain = cryptoHelpers.convertByteArrayToString(decrypt);
87   if((bytesToDecrypt.length % 16) != 0) {
88     plain = plain.substr(0, plain.length-1);
89   }
90   //console.log(cypher + ";" + key + ";" + plain);
91   return plain;
92 }
93
94
95 //Request Login to Web Service
```

```
96  function requestLogin (username, password, callback, fallback) {
97    ajaxRequest.onreadystatechange = function() {
98      if(ajaxRequest.readyState == 4) {
99        if(ajaxRequest.status == 200) {
100         log("RESPONSE", "LOGIN");
101
102         /** Check Syntax of JSON Response **/
103         var jsonResponse;
104         //console.log(ajaxRequest.responseText);
105         try {
106           jsonResponse = JSON.parse(ajaxRequest.responseText);
107           if (
108             typeof jsonResponse.STATUS === "undefined"    ||
109             typeof jsonResponse.ID === "undefined"        ||
110             typeof jsonResponse.EKsT === "undefined"      ||
111             typeof jsonResponse.Sa === "undefined"        ||
112             typeof jsonResponse.Sb === "undefined"        ||
113             typeof jsonResponse.signature === "undefined"
114           ) {
115             var isServerAvaiable = changeServer();
116             if(isServerAvaiable) {
117               requestLogin(username, password, callback, fallback);
118             }
119             else {
120               fallback("There is no Web Server avaiable! Please try
                      again later.");
121             }
122           }
123         }
124         catch (e) {
125           var isServerAvaiable = changeServer();
126           if(isServerAvaiable) {
127             requestLogin(username, password, callback, fallback);
128           }
129           else {
130             fallback("There is no Web Server avaiable! Please try again
                    later.");
131           }
132         }
133
134         /** Check Signature **/
135         var signature = jsonResponse.signature;
136         var transform = jsonResponse.STATUS + jsonResponse.ID +
              jsonResponse.EKsT + jsonResponse.Sa + jsonResponse.Sb;
137         var isValid = doVerify(transform, signature);
138         /** Signature OK! **/
139         if(isValid) {
140           /** Check STATUS, ID and Sa **/
141           if(jsonResponse.STATUS == "OK") {
142             if (jsonResponse.Sa == Sa && jsonResponse.ID == ID) {
143               //Set the Session Key
144               Ks = hex_sha1(jsonResponse.Sa + jsonResponse.Sb + P);
145               //Set the Token
146               T = aesDecode(jsonResponse.EKsT, Ks.substr(0,16));
147               console.log(T);
148               //Show Initial Page
```

```
149             callback();
150           }
151           // Sa and/or ID wrong!
152           else {
153             var isServerAvaiable = changeServer();
154             if(isServerAvaiable) {
155               requestLogin(username, password, callback, fallback);
156             }
157             else {
158               fallback("There is no Web Server avaiable! Please try
                      again later.");
159             }
160           }
161         }
162         // STATUS == "NOK"
163         else {
164           var isServerAvaiable = changeServer();
165           fallback("There was a problem in the contact with one of
                ours servers.<br> Please, login again.");
166           if(!isServerAvaiable) {
167             currentServer = -1;
168             changeServer();
169           }
170         }
171       }
172       /** Signature NOK! **/
173       else {
174         var isServerAvaiable = changeServer();
175         if(isServerAvaiable) {
176           requestLogin(username, password, callback, fallback);
177         }
178         else {
179           fallback("There is no Web Server avaiable! Please try again
                later.");
180         }
181       }
182     }
183     /** HTTP Status != 200 **/
184     else {
185       var isServerAvaiable = changeServer();
186       if(isServerAvaiable) {
187         requestLogin(username, password, callback, fallback);
188       }
189       else {
190         fallback("There is no Web Server avaiable! Please try again
              later.");
191       }
192     }
193   }
194 };
195
196 /** PREPARE **/
197 // Set the Global Variable ID
198 ID = username;
199
200 // The Secret's Hash
```

```
201   P = hex_sha1(password);
202   console.log(P);
203   // The Client's Salt
204   var Sa = Math.floor((Math.random()*1e+10)+1);
205
206   // The Authentication's Hash Field
207   var h = hex_sha256(P + Sa);
208
209
210   /** SEND **/
211   var message = "ID=" + ID + "&Sa=" + Sa + "&h=" + h;
212   var url = baseUrl + "/Login?" + message;
213   //console.log(url);
214   ajaxRequest.open("get", url, true);
215   log("REQUEST", "LOGIN");
216   ajaxRequest.send(null);
217 }
218
219 //Request Data to Web Service
220 function requestData(data, urlRequest, callback, fallback,
      transformJsonFunction) {
221   ajaxRequest.onreadystatechange = function() {
222     if(ajaxRequest.readyState == 4) {
223       if(ajaxRequest.status == 200) {
224
225         log("RESPONSE", urlRequest);
226         //console.log(ajaxRequest.responseText);
227         var jsonResponse;
228
229         /** Sintaxe Validation **/
230         try {
231           jsonResponse = JSON.parse(ajaxRequest.responseText);
232           if (
233             typeof jsonResponse.Status === "undefined"    ||
234             typeof jsonResponse.ID === "undefined"        ||
235             typeof jsonResponse.TOp === "undefined"      ||
236             typeof jsonResponse.RS === "undefined"        ||
237             typeof jsonResponse.EKsT === "undefined"        ||
238             typeof jsonResponse.signature === "undefined"
239           ) {
240             var isServerAvaiable = changeServer();
241             if(isServerAvaiable) {
242               requestData(data, urlRequest, callback, fallback,
                     transformJsonFunction);
243             }
244             else {
245               fallback("There is no Web Server avaiable! Please try
                     again later.");
246             }
247           }
248         }
249         catch (e) {
250           var isServerAvaiable = changeServer();
251           if(isServerAvaiable) {
252             requestData(data, urlRequest, callback, fallback,
                   transformJsonFunction);
```

```
253           }
254           else {
255             fallback("There is no Web Server avaiable! Please try again
                  later.");
256           }
257         }
258
259         /** Check Signature **/
260         var signature = jsonResponse.signature;
261         var transform = "";
262         if(transformJsonFunction != null) {
263           transform = transformJsonFunction(jsonResponse.RS);
264         }
265         else {
266           transform = jsonResponse.RS;
267         }
268         var messageToVerify = jsonResponse.Status + jsonResponse.ID +
              jsonResponse.TOp + transform + jsonResponse.EKsT;
269         var isValid = doVerify(messageToVerify, signature);
270         if(isValid) {
271           /** Status Validation **/
272           if(jsonResponse.Status == "OK") {
273             /** ID and T Validation **/
274             if(jsonResponse.ID == ID && jsonResponse.TOp == TOp) {
275               T = aesDecode(jsonResponse.EKsT, Ks.substr(0,16));
276               console.log(T);
277               callback(jsonResponse.RS);
278             }
279             else {
280               var isServerAvaiable = changeServer();
281               if(isServerAvaiable) {
282                 requestData(data, urlRequest, callback, fallback,
                      transformJsonFunction);
283               }
284               else {
285                 fallback("There is no Web Server avaiable! Please try
                      again later.");
286               }
287             }
288           }
289           // Status Invalid
290           else {
291             console.log("NOK;Integrity Invalid");
292             var isServerAvaiable = changeServer();
293             if(isServerAvaiable) {
294               fallback("There was a problem in the contact with one of
                    ours servers.<br> Please, login again.");
295             }
296             else {
297               fallback("There is no Web Server available.<br> Please,
                    try again later.");
298             }
299           }
300         }
301         // Integrity Invalid
302         else {
```

```
303           console.log(jsonResponse.Status);
304           var isServerAvaiable = changeServer();
305           if(isServerAvaiable) {
306             requestData(data, urlRequest, callback, fallback,
                  transformJsonFunction);
307           }
308           else {
309             fallback("There is no Web Server avaiable! Please try again
                  later.");
310           }
311         }
312       }
313       // Timeout
314       else {
315         var isServerAvaiable = changeServer();
316         if(isServerAvaiable) {
317           requestData(data, urlRequest, callback, fallback,
                transformJsonFunction);
318         }
319         else {
320           fallback("There is no Web Server avaiable! Please try again
                later.");
321         }
322       }
323     }
324   };
325
326   /** PREPARE **/
327   // The Operartion Token encrypted
328   var TOp = (parseInt(T) + 1).toString();
329
330   // The message
331   var message = "ID=" + ID + "&TOp=" + TOp + "&D=" + data;
332
333   // The Integrity's Field
334   var transform = ID + TOp + data;
335   var mac = hex_sha256(transform + Ks);
336   message += "&MAC=" + mac;
337
338   /** SEND **/
339   var url = baseUrl + urlRequest + message;
340   ajaxRequest.open("get", url, true);
341   log("REQUEST", urlRequest);
342   ajaxRequest.send(null);
343 }
344
345
346
347 //Log REQUEST and RESPONSE events
348 function log(type, operation) {
349   var currentdate = new Date();
350   var datetime = currentdate.getDate() + "/"
351                  + (currentdate.getMonth()+1)  + "/"
352                  + currentdate.getFullYear() + " @ "
353                  + currentdate.getHours() + ":"
354                  + currentdate.getMinutes() + ":"
```

```
355                     + currentdate.getSeconds() + "."
356                     + currentdate.getMilliseconds();
357     console.log(type + " to Operation " + operation + " in " + datetime);
358 }
359
360 client.send()
```

Abreviaturas

AJAX Assynchronous JavaScript and XML. 2

API Application Programming Interface. 4

CBC Cipher Block Chaining. 47

CGI Common Gateway Interface. 3

CORS Cross-Origin Resource Sharing. 30

CPU Central Process Unit. 50

DOM Document Object Model. 2

HTML Hyper Text Markup Language. 2

HTTP Hypertext Transport Protocol. 2

JAR Java Archive. 20

JSON JavaScript Object Notation. 2

LAN Local Area Network. 9

MAC Message Authentication Code. 19

PGP Pretty Good Privacy. 21

RAM Random Access Memory. 3

RMI Remote Method Invocation. 44

SOMA Same Origin Mutual Approval. 11

TRIALs Trusted Rich Internet Application Layers. 1

URL Uniform Resource Locator. 20

XML Extensible Markup Language. 2

Bibliografia

[1] *Secure hash standard*. National Institute of Standards and Technology, Washington, 2002. URL: `http://csrc.nist.gov/publications/fips/`. Note: Federal Information Processing Standard 180-2.

[2] Christian Cachin abd Rachid Guerraoui and Luís Rodrigues. *Introduction to Reliable and Secure Distributed Programming*, page 29. Springer, Second edition, 2011.

[3] Eduardo Adilio Pelinson Alchieri, Alysson Neves Bessani, and Joni da Silva Fraga. A dependable infrastructure for cooperative web services coordination. In *Proceedings of the 2008 IEEE International Conference on Web Services*, ICWS '08, pages 21–28, Washington, DC, USA, 2008. IEEE Computer Society.

[4] Julian Aubourg, Jungkee Song, and Hallvord Steen. Xmlhttprequest w3c working draft 6 december 2012. `http://www.w3.org/TR/XMLHttpRequest/`, June 2013.

[5] T. Aura. Strategies against replay attacks. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 59–68, 1997.

[6] A. Avižienis and L. Chen. On the implementation of N-version programming for software fault tolerance during execution. In *Proceedings of the IEEE International Computer Software and Applications Conference*, pages 149–155, 1977.

[7] Alysson Neves Bessani, Eduardo Pelison Alchieri, Miguel Correia, and Joni Silva Fraga. Depspace: a byzantine fault-tolerant coordination service. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, pages 163–176, New York, NY, USA, 2008. ACM.

[8] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). `http://www.w3.org/TR/REC-xml/`, June 2013.

[9] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[10] Stephen Chong, K. Vikram, and Andrew C. Myers. Sif: enforcing confidentiality and integrity in web applications. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 1:1–1:16, Berkeley, CA, USA, 2007. USENIX Association.

[11] D. Crockford. RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON). Technical report, IETF, 2006.

[12] Charlie Curtsinger, Benjamin Livshits, Benjamin Zorn, and Cristian Seifert. ZOZZLE: Fast and precise in-browser JavaScript malware detection. In *Proceedings of the USENIX Annual Technical Conference '11*, Portland, USA, June 2011.

[13] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, Berlin, Heidelberg, New York, 2002.

[14] Tobias Distler, Ivan Popov, Wolfgang Schröder-Preikschat, Hans P. Reiser, and Rüdiger Kapitza. Spare: Replicas on hold. In *NDSS*. The Internet Society, 2011.

[15] Java SE Documentation. JDK 6 Java Database Connectiviti (JDBC) - related API and Developer Guides. `http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/`, June 2013.

[16] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[17] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro. Analysis of OS diversity for intrusion tolerance. *Software: Practice and Experience*, 2013.

[18] Jesse Garret. Ajax: A new approach to web applications. `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications`, June 2013.

[19] William Grosso. *Java RMI*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 2001.

[20] Google Project Hosting. slowaes - scripting language open-source workable aes. `http://code.google.com/p/slowaes/`, June 2013.

[21] Ecma International. Standard ECMA-262 ECMAScript Language Specification. Technical report, Ecma International, 2011.

[22] B. Kaliski. Public-key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, 2003.

[23] RSA Laboratories. What is Cipher Block Chaining Mode? http://www.rsa.com/rsalabs/node.asp?id=2171, June 2013.

[24] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[25] Nigel McFarlane. Script-based security. http://msdn.microsoft.com/en-us/library/ms970704.aspx, June 2013.

[26] David Mosberger and Tai Jin. httperf - a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, December 1998.

[27] Alexander Moshchuk, Tanya Bragin, Damien Deville, Steven D. Gribble, and Henry M. Levy. SkyProxy: Execution-based detection of malicious web content. In *Proceedings of 16th USENIX Security Symposium*, Berkeley, USA, September 2007.

[28] Myers M, et. al. Internet Engineering Task Force. RFC2560 X.509 Internet Public Key Infrastructure, Online Certificate Status Protocol, 1999.

[29] Terri Oda, Glenn Wurster, P. C. van Oorschot, and Anil Somayaji. SOMA: Mutual approval for included content in web pages. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, New York, USA, October 2008.

[30] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[31] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. Stronger password authentication using browser extensions. In *Proceedings of the 14th conference on USENIX Security Symposium*. USENIX Association, 2005.

[32] Jesse Ruderman. Same origin policy for javaScript. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Same_origin_policy_for_JavaScript, June 2013.

[33] Jesse Ruderman. Signed script in mozilla. http://www.mozilla.org/projects/security/components/signed-scripts.html, June 2013.

[34] William Stallings. *Cryptography and Network Security Principles and Practice*, pages 395–398. Pearson, Fifth edition, 2011.

[35] Kenji Urushima. jsrsasign - rsa signing and verification in javascript. http://kjur.github.io/jsrsasign/, June 2013.

[36] Anne van Kesteren. Cross origin resource sharing. `http://www.w3.org/TR/cors`, June 2013.

[37] Paulo Veríssimo and Nuno Ferreira Neves (editors). Service and protocol architecture for the maftia middleware, 2001.

[38] Paulo Esteves Veríssimo, Nuno Ferreira Neves, and Miguel Pupo Correia. Architecting dependable systems. chapter Intrusion-tolerant architectures: concepts and design, pages 3–36. Springer-Verlag, Berlin, Heidelberg, 2003.

[39] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. Ebawa: Efficient byzantine agreement for wide-area networks. In *HASE*, pages 10–19. IEEE Computer Society, 2010.

[40] Paulo Veríssimo and Luís Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publisher, First edition, 2001.

[41] Lidong Zhou, Fred B. Schneider, and Robbert Van Renesse. Coca: A secure distributed online certification authority. *ACM Trans. Comput. Syst.*, 20(4):329–368, November 2002.

[42] Lidong Zhou, Fred B. Schneider, and Robbert Van Renesse. Apss: proactive secret sharing in asynchronous systems. *ACM Trans. Inf. Syst. Secur.*, 8(3):259–286, August 2005.